

Sumário

1. O que é o Python?	2
1.1 Linguagem de Programação	2
1.2 Algoritmos	2
2. Como começar a programar?	4
2.1 IDE	4
3. Operadores Aritméticos	5
3.1 Import Math - Avançando nas operações matemáticas	6
4. Variáveis	9
4.1 Tipos de Variáveis	10
4.2 Dados de entrada e saída	11
4.3 Exercícios	13
5. Condicionais	14
5.1 If, Else e Elif	14
5.2 Operadores de comparação	15
6. Indentação	16
7. Loop	17
7.1 for	17
7.2 while	18
8. Funções	20
8.1 Definindo funções	20
8.2 Funções lambda	21
8.3 Função Round	22
9. Bibliotecas	23
9.1 Matplotlib	23
9.2 Numpy	25
9.3 Pandas	27
10. Adorei o Python, posso virar um profissional de programação só com ele?	29
11. Exercícios	29
11.1 Resolução	31
12. Conclusão	34

Documento de Suporte - Python

Tudo bem, atenderemos ao seu pedido de socorro! 👍

Escrito por Guilherme Xavier, Victória Barros, Alan Welisson, Ana Clara Ruiz, Nicolas Macedo e Bernardo Lopes em 19/08/2023 - Todos os direitos reservados.

Atualizado por Mariana Fonseca em 09/03/2024.

1. O que é o Python?

Python é uma **linguagem de programação** bem simples, quando comparada a outras, apresentando uma sintaxe tranquila - alguns brincam que você escreve inglês com o computador - e com uma ótima abstração. Isso é como dizer que você vai dirigir um carro automático, onde muitas coisas, como a troca de marchas - que é uma necessidade do automóvel - é abstraída de você, assim, você não precisa se preocupar com isso.

O Python abstrai muitas necessidades do computador de nós, fazendo com que possamos nos preocupar com menos informações, que - assim como dirigir um carro - parece nos sufocar.

1.1 Linguagem de Programação

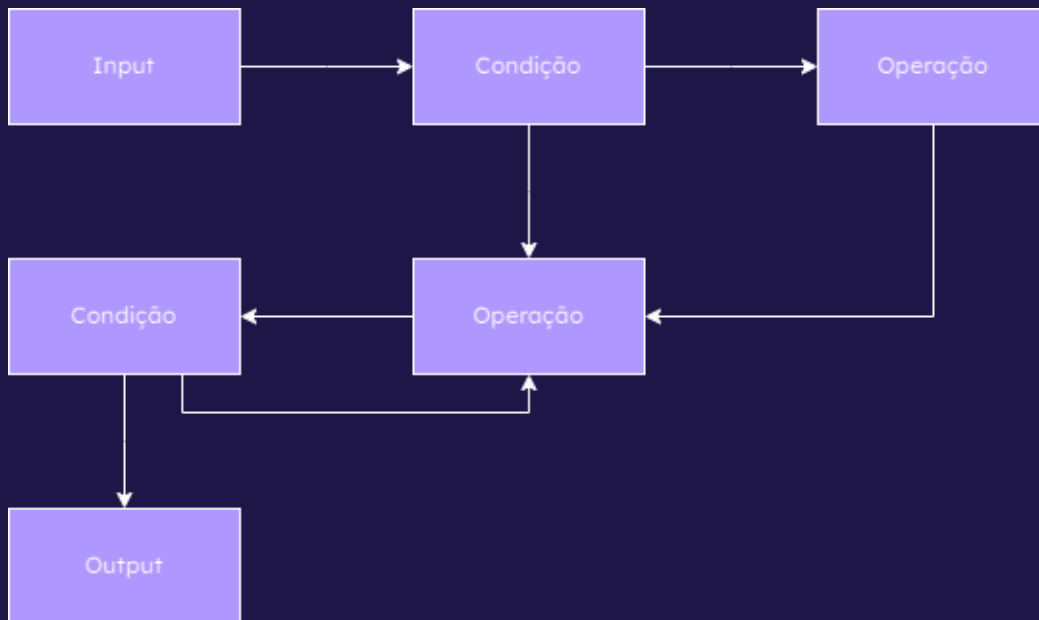
Uma linguagem de programação possui essa nomenclatura porque permite que você desenvolva **fluência na comunicação com o computador**. Quanto mais profundamente você se aventurar no aprendizado de linguagens, mais se aproxima da compreensão das engrenagens internas do mundo da computação. Este documento marca o ponto de partida nessa jornada, com a introdução à linguagem Python. Ele é uma **excelente escolha** para iniciar, devido à sua simplicidade e versatilidade, o que o torna uma porta de entrada acessível para o vasto universo da programação.

1.2 Algoritmos

Quando o computador roda um código, ele está seguindo uma série de instruções que o programador escreveu. Essas instruções compõem um **algoritmo**,

for_code[]

que é o equivalente a uma receita para o computador. Os algoritmos podem ser representados por etapas ou por fluxogramas.



Exemplo de fluxograma de um algoritmo

O exemplo acima é bem genérico, mas a ideia é mostrar que o código segue um fluxo de instruções. O algoritmo começa com a entrada de dados, os quais podem passar por operações, condições e/ou repetições, e no caso acima, o input passa por uma condição. Dependendo se o dado cumpre ou não a condição, ele pode ter que passar por um processo antes de seguir para a próxima operação. Depois disso, há mais uma condição, e, caso não seja satisfeita, o dado precisa repetir a operação anterior até cumprir a exigência, e por fim um output é produzido.

Antes de aprender a programar, é importante aprender sobre **Lógica de Programação**, ou seja, a maneira como estruturar um algoritmo. O formato do algoritmo pode ser feito de infinitas maneiras dependendo do propósito do código, e para elaborá-lo são usadas ferramentas como **variáveis, operadores, estruturas de condição e repetição, vetores/listas e matrizes**. Estas ferramentas serão abordadas mais adiante, e ao longo deste documento é importante entender os conceitos delas para que você possa planejar bem seus códigos.

2. Como começar a programar?

2.1 IDE

Para começar a programar, somente é preciso de uma IDE e, em alguns casos, da instalação do python. Para a feitura do nosso case, propomos o uso da Google Colab que não precisará da instalação do python como será explicado abaixo.

- Editor de Texto Maneiro:

Também chamado de **IDE**, que significa **ambiente integrado de desenvolvimento**, um código pode ser feito no Word ou até em algum bloco de notas, mas ter uma IDE facilita o trabalho.

Nós indicamos, para esse Processo Seletivo, o **Google Colab** ou **Colaboratory**. É um site sem custos financeiros da Google Research que permite escrever e executar Python dentro do navegador, já contando com: acesso a GPUs sem custo financeiro, compartilhamento fácil e sem necessidade de configuração.



Imagem 1: Ícone do Google Colab

- Como utilizar:

Depois de acessar o link abaixo, deve-se logar em uma conta google para começar a programar. A partir deste ponto, é só ir em arquivo → novo notebook e, então, tudo está pronto para começar a fazer códigos.

[Página Inicial do Colab](#)

O Colab é um serviço de notebooks hospedados do Jupyter. Dentro do mesmo, temos um ambiente interativo, que permite escrever e executar código, podendo ter dois tipos de célula: código e textuais (formato markdown). Assim, o notebook do Colab permite combinar códigos e rich text em um só documento, além de imagens, HTML, LaTeX, entre outros.

for_code[]

Dentro do notebook, as marcações do canto superior esquerdo +Code e +Text adicionam, respectivamente, as células de código e de texto.

As células de código são executadas utilizando o botão Play à esquerda do código ou o atalho “Command/Ctrl+Enter”. Já para editar, clicamos na célula e começamos a editar.

- Armazenamento e Compartilhamento:

Além disso, o Colab é armazenado na sua conta do Google Drive ou pode ser carregado pelo GitHub, podendo ser compartilhado com outros usuários de forma fácil, permitindo-os editar, comentar ou somente ler o notebook.

Também é possível fazer upload de notebooks do Jupyter/IPython para o Colab e fazer download do arquivo para enviar para pessoas que queiram visualizar o mesmo por outra IDE, baixando-os como arquivo .ipynb (notebook de código aberto do Jupyter).

3. Operadores Aritméticos

Uma linguagem de programação possui algumas **estruturas-chave** que são necessárias para que você consiga utilizá-la para resolver seus problemas. Para demonstrar cada uma, eu boleei um esquema reverso: vou te dar o problema e depois te mostrar como cada estrutura resolve esse problema.

Primeiro problema:

As utilizações do computador giram em torno de **fazer contas** com ele. Imagine só nosso mundo sem calculadoras... Iríamos voltar ao século passado em uma tacada só. Então, precisamos aprender como fazemos conta com o computador.

No Python, temos os seguintes **operadores**:

Operador soma (+) → $3 + 3 = 6$

Operador subtração (-) → $3 - 3 = 0$

Operador multiplicação (*) → $3 * 3 = 9$

Operador divisão fracionária (/) → $3 / 3 = 1$ ou $5 / 2 = 2.5$

Operador divisão inteira (//) → $3 // 3 = 1$ ou $5 // 2 = 2$

for_code[]

Operador potenciação ($$)** $\rightarrow 3 ** 3 = 27$

Operador resto da divisão ($\%$) $\rightarrow 3 \% 3 = 0$ ou $7 \% 3 = 1$

É importante ressaltar que esses operadores seguem a exata mesma ordem que a matemática segue:

Parêntesis, potenciação e radiciação, multiplicação e divisão, soma e subtração.

Esses são todos os operadores aritméticos que existem no Python, mas, como você deve ter percebido, **faltam** várias coisas para que possamos fazer todas as contas: **Raízes, seno, logaritmos e mais.**

No caso de raízes, até que dá pra fazer usando exponenciação em números fracionários do tipo $3**(1/2)$, porém, seno e logaritmo, por exemplo, são mais complicados.

Por isso, existem **extensões**, do próprio Python ou externas, que são pedaços de código que outras pessoas já fizeram para te facilitar e adicionam novas funcionalidades para que possamos utilizar no nosso código. Essas extensões também são conhecidas como **bibliotecas**. Segura a ansiedade, porque teremos uma sessão inteira sobre elas!

3.1 Import Math - Avançando nas operações matemáticas

Tá, prosseguindo com o nosso tópico anterior, sabemos fazer operações básicas com Python, mas como faço cosseno então? Que biblioteca externa é essa?

Como diz nosso atual Vice-Presidente Bernardo: "fique calmo, meu pequeno gafanhoto! É muito simples":

Para **importar um módulo externo** é só utilizar (por convenção no início do código) da palavra reservada **import**. Nesse caso, iremos importar uma biblioteca feita pelo próprio Python, que se chama **math**. Então, trate de escrever o comando em seu código teste:

```
import math
```

Esse comando irá importar todo esse bloco de código, que trará algumas funções com ele. Uma delas, que você pode testar é o `math.cos(valor)` que você pode testar facilmente também no seu código.

for_code[]

Algumas noções importantes:

- Python é *case-sensitive*, isso significa que escrever **Import** é diferente de **import**. Fique atento ao uso de **maiúsculas e minúsculas**.
- Quando se importa uma parte de código dessa forma, normalmente você utiliza o operador ponto "." para acessar o que está dentro dele, seguido da função - que é um pequeno bloco de código escrito por alguém.

Ex.: `math.cos(valor)` → `math` é o bloco de código. Puxei a função `cos` de dentro dele e pedi pra ela calcular o `valor` que eu quero.

- Todas as funcionalidades de cada bloco de código estão escritas em sua respectiva **documentação**, que é uma espécie de manual de instruções de como funciona o bloco de código que foi programado. Os bons programadores que eu conheço sempre lêem toda a documentação das tecnologias que eles utilizam. 😊

Bom, essa é a documentação da biblioteca `math`, que você acabou de importar (Se você der um `Ctrl + F` para pesquisar e jogar um `math.cos(x)` lá, você vai encontrar o comando que comentei, que retorna nosso cosseno):

<https://docs.python.org/3/library/math.html>

É uma documentação bem amigável. Pode servir como um dicionário, onde, quando você precisa de alguma funcionalidade específica, dá uma lida nela e procura rapidamente como se faz.

Se isso tudo está meio confuso em sua cabeça, fique tranquilo que nós já iremos aplicar isso tudo e por a mão na massa. 😊

Terceiro problema:

Tá, tudo bem. Agora eu quero ver o resultado do cosseno na minha tela, como vou saber se ele está calculando o cosseno mesmo?

Verdade, é legal vermos se estamos realmente fazendo progresso, né? Tá, vou ensinar pra você uma built-in function (funções já embutidas no Python) que podem te ajudar com isso tudo: (Lembre-se que maiúsculas e minúsculas mudam todo o comando. `Print` e `print` são distintos).

`print(valor)` → Retorna o que for passado dentro dele (Já vai ficar mais claro).

Agora já dá pra fazer algo interessante, no seu código, veja se faz sentido todas as expressões que te expliquei com esse código aqui:

for_code[]

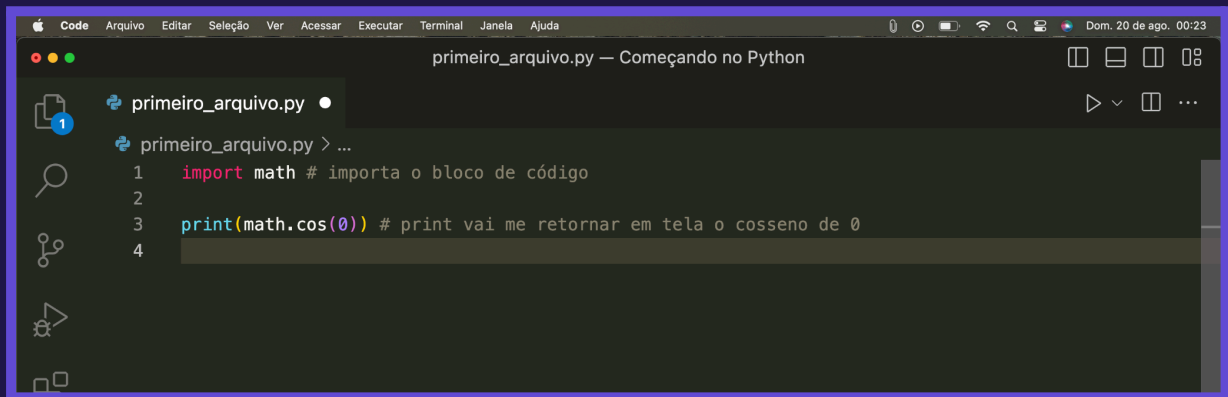


Imagem 4: Exemplo de valor para retornar

Vale dizer também, que quando se coloca o jogo da velha (#) em uma linha, todo o conteúdo depois dela é ignorado, isso é utilizado para comentar o funcionamento do que você está criando para se lembrar ou para instruir outras pessoas de como seu código está funcionando. É uma ótima prática a se utilizar.

Continuando, para fazer a magia acontecer, pode clicar no play, lá em cima.

E olhe bem o nosso resultado:

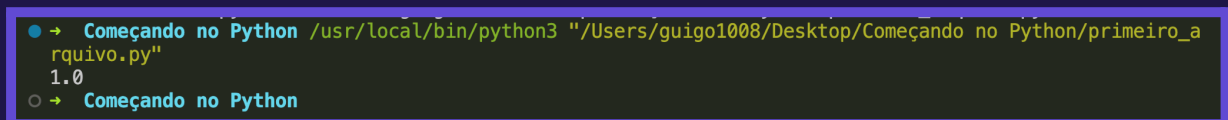


Imagem 5: Output da função print

Ainda bem que o cosseno de 0 deu 1, Ufa! Vencemos! 🎉

Quarto problema:

Show, estamos indo bem, mas você concorda que se fosse um código enorme, com muito mais repetição, fazer assim seria muito mais difícil?

```
import math # importa o bloco de código

# vamos achar as raízes de x**2 + 2x + 1
print((-2 + math.sqrt(2**2 - 4 * 1 * 1)) / 2*1)
print((-2 - math.sqrt(2**2 - 4 * 1 * 1)) / 2*1)
# conta grande e confusa - Bháskara
```


for_code[]

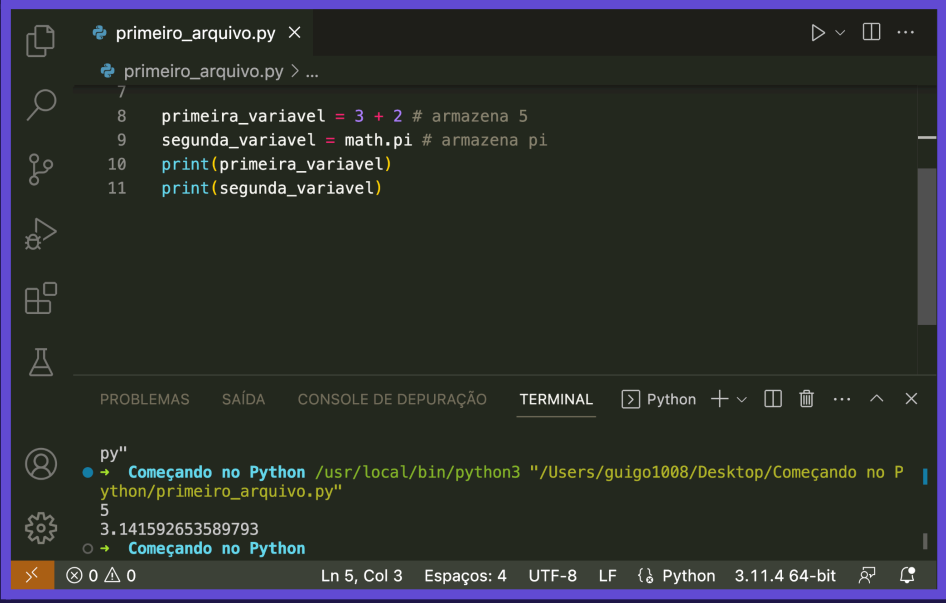
Imagem 6: Fórmula de Bháskara

Esse é só um exemplo tranquilo ainda por cima. Enfim, te convenci que precisamos dar uma organizada melhor nesse código, portanto vou te introduzir o conceito de variáveis.

4. Variáveis

Você prefere ter que **decorar** todos os seus contatos, para ter que **discar o número** deles de cor toda vez que tiver que usar ou criar uma **variável** com o **nome da pessoa**, para simplesmente escrever o nome dela e a memória do celular discar o número para você?

Nem precisa responder. Vamos aprender como se cria variáveis. É muito simples, coloque o nome da variável adicione um igual (=) depois dela e logo em seguida pode escrever o que quer armazenar. Quer exemplos?



```
primeiro_arquivo.py X
primeiro_arquivo.py > ...
7
8 primeira_variavel = 3 + 2 # armazena 5
9 segunda_variavel = math.pi # armazena pi
10 print(primeira_variavel)
11 print(segunda_variavel)

PROBLEMAS SAÍDA CONSOLE DE DEPURACÃO TERMINAL Python + - ... ^ X

py"
→ Começando no Python /usr/local/bin/python3 "/Users/guigo1008/Desktop/Começando no P
ython/primeiro_arquivo.py"
5
3.141592653589793
○ → Começando no Python

Ln 5, Col 3 Espaços: 4 UTF-8 LF Python 3.11.4 64-bit
```

Imagem 7: Atribuição de valores à variáveis

Muito melhor, nós **guardamos o valor em cada variável** e **acessamos esses valores por nomes**, o que é muito mais fácil de se gravar. Vamos melhorar aquele Bháskara também? (Deixei um errinho bobo de propósito pra você me falar o que está errado.)

for_code[]

```
import math # importa o bloco de código

# vamos achar as raízes de x**2 + 2x + 1
raiz_de_delta = 2**2 - 4 * 1 * 1
# variavel que ja guarda o valor da raiz de delta

print((-2 + raiz_de_delta) / 2*1)
print((-2 - raiz_de_delta) / 2*1)
```

Imagem 8: Fórmula de Bháskara com o uso de variáveis

Ficou bem mais fácil e simples de se enxergar, concorda? Agora vamos para alguns detalhes e algumas regras de como se pode ou não fazer variáveis:

- É proibido criar variáveis:

1. Que **não** comecem com uma letra ou "underline" (_). Ex.: 1variavel, *variavel.
2. Que possuam caracteres diferentes de números, "underline" ou letras. Ex.: primeira variavel. Prefira criar primeira_variavel.
3. Que tenham nomes já utilizados no Python para alguma coisa. Ex.: print. O print já é um nome reservado para a função print do Python.

Exemplos de variáveis permitidas:

```
n1 = 2 + 5
```

```
teste_1 = math.sqrt(4)
```

```
for_code = "melhor liga de computação da UFRJ"
```

```
coração = 5**2
```

**Você pode utilizar acentos e cedilhas no nome da variável, mas é uma boa prática evitar.*

É uma boa prática **colocar nomes que te remetem a algo na variável**. Não seja o programador que chama suas variáveis de v1, v2, v3, v4 porque você vai eventualmente se perder no meio disso. Você gostaria que eu mudasse o nome dos seus contatos para c1, c2, c3, c4 ?

4.1 Tipos de Variáveis

Você não achou que nós poderíamos guardar somente números nessas variáveis, certo? Se achou, estava errado. Vou te mostrar tudo que nós podemos guardar:

for_code[]

1. **int** (inteiro): 1 → Somente números inteiros.
2. **float** (ponto flutuante): 5.6 → Números com vírgula.
3. **string**: "bom dia!" → Textos.
4. **Booleano**: True ou False → É muito importante, são geradas quando se faz alguma comparação dentro do Python. Também ficará mais claro depois
5. **lista**: [1,"dois", 3.4, "banana", "pizza"] → Armazena uma lista de elementos. Note que sempre tem os colchetes ([]) em torno da lista e você pode armazenar diferentes tipos dentro dela, como strings, inteiros e mais.
6. **tupla**: (5, "coisa", -7.18, 5e+15) → É parecida com uma lista, mas a diferença é que uma tupla não pode ser alterada, ou seja, seu conteúdo é imutável.
7. **dicionário**: {"Adriano": 98765-4321, "Bernardo": 99999-9999} → Também parece bastante com a variável de tipo lista, porém cada item da lista é nomenclaturado. Quando usarmos, mais em breve ficará mais claro.

Você pode guardar textos em variáveis, mas esses textos são chamados de string. Para armazenar, a condição é que o texto tem que estar envolvido de aspas (") ou apóstrofos ('). Se você escrever eles sem esses caracteres que eu comentei, você está copiando o conteúdo de uma variável em outra.

Tipo de variável	Valor
String(str)	"Eu amo programar"
Inteiro (int)	1, 3, -10, 0, 100
Ponto flutuante (float)	3,14, 2.7, 0.5
Booleano (Bool)	True, False

4.2 Dados de entrada e saída

Você também pode inserir qualquer um dos tipos de variáveis apresentados através do próprio prompt, chamamos isso de **entrada de dados**. O comando mais utilizado é o **input**, que funciona como entrada de valores numa variável com seu tipo já estabelecido (textos em variáveis string e inteiros em variáveis int). Para a

for_code[]

saída de dados, ou seja, o **output**, o comando mais simples é o `print`, que imprime o valor de uma variável no prompt.

Exemplo 1:

```
nome = str(input("Insira seu nome"))
```

```
nota = float(input("Qual sua nota?"))
```

Para a saída de dados, o comando mais simples é o `print`, que imprime algum valor ou variável no prompt.

Exemplo 2:

```
print(nome)
```

```
print('programar é minha paixão!')
```

Exemplo 3:

Lembre-se de sempre declarar o tipo de variável antes de escrever o `input` senão ocorrerá o seguinte erro:

```
x = input('insira um número')
```

```
y = input('insira outro número')
```

```
print(x + y)
```

Se você escreveu 2 em x e 3 em y, sua resposta não será 5 como o esperado, mas sim 23. Percebeu o erro? O Python entende as duas variáveis como strings e então une as duas, por isso precisamos colocar o tipo de variável como `int` para que os valores inseridos sejam reconhecidos como números inteiros

Exemplo 4:

`variavel_1 = texto` (Você está armazenando o conteúdo da variável **texto** dentro da **variavel_1**, para ser string seria: `"texto"` ou `'texto'`).

Não se preocupe. Só lembre-se que essa parte do material existe para que você possa consultar mais em breve.

4.3 Exercícios

Beleza, vamos tentar brincar com que aprendemos então. Consegue me entregar esses códigos?

1. Crie um código que calcule a tangente de pi.
2. Agora um código que calcule o $\log_2(8)$.
3. Você pode fazer um código que calcule sua média em uma matéria.
4. E um que converte graus em radianos.

Muito bom, espero que tenha conseguido. Caso sinta dificuldade, recomendo novamente, que entre na documentação da biblioteca math para descobrir como se faz cada exercício.

Link documentação math: <https://docs.python.org/3/library/math.html>

Aliás, vou até te dar uma ajudada... Eu pesquisei sobre o logaritmo na base 2 e achei isso aqui. Te ajuda?

```
math.log2(x)
Return the base-2 logarithm of x. This is usually more accurate than log(x, 2).

New in version 3.3.

See also: int.bit_length() returns the number of bits necessary to represent an integer in binary, excluding the sign and leading zeros.
```

Imagem 9: Descrição da função math.log2()

Vou te mostrar como eu leria essa documentação, se você leu e entendeu tudo, pode pular essa parte.

Seguinte, ele está usando aquela função `math.log2(valor)` e isso vai me retornar o valor do \log_2 do número que eu quiser colocar. Neste caso, vou colocar 8 aqui mesmo:

$\log_2(8)$

Outra coisa que ele falou é que fazer isso, geralmente é mais acurado do que utilizar o `log(valor, 2)` que é outro comando para qualquer logaritmo onde a base é o segundo valor e o primeiro valor é o número em questão que se quer colocar.

Então, ponha as mãos na massa se você ainda não conseguiu fazer aqueles exercícios e trate de fazer.

5. Condicionais

Quer mais problemas? Pode deixar comigo. Quero que você tente fazer agora um código mais elaborado:

Você vai escrever um código, agora, que tenha valores de 0 a 10 em variáveis chamadas: `prova_1` e `prova_2`. Com esses valores, calcule a média deles e imprima em tela 0 para reprovado ($\text{Nota} < 5$) e 1 para aprovado ($\text{Nota} \geq 5$). Óbvio que, você fez o exercício que eu te mandei, então você pode aproveitar a parte que você já feita. (Como diria um grande professor que eu tive: "Eu não dou ponto sem nó!").

Tente fazer. Estou esperando... Não existe teoria sem prática.

Descobriu a dificuldade? Com o atual conhecimento que nós temos, não dá pra dividir o fluxo do código. Bifurcar o que ele vai imprimir em tela em dois casos possíveis. Tente fazer o exercício que vai ficar mais claro. Ok, dito isso, vou te ensinar a dividir o fluxo do código. Vamos lá.

5.1 If, Else e Elif

Existe as seguintes palavras reservadas dentro do Python para isso:

```
if (condição):
```

```
    print(1)
```

```
else:
```

```
    print(0)
```

Isso significa que:

Se (`if`) a condição colocada dentro do parêntesis for verdadeira, execute o código dentro desse escopo. Ou seja, o comando `if` é utilizado para **verificar uma expressão e executar um bloco de código**, caso ela seja verdadeira. Caso essa expressão seja **falsa**, utilizamos o comando `else`. Esse comando basicamente diz: siga por esse **bloco de código alternativo**. Também temos o comando `elif`, que é utilizado quando mais de uma condição de `if` precisa ser testada.

Porém, temos que entender que condição é essa, certo? Vou te mostrar como funcionam as condicionais no Python.

5.2 Operadores de comparação

Digamos que estamos comparando uma variável **x** com outra **y**. Para determinar se uma condição é verdadeira ou falsa, utilizamos os **operadores de comparação**. Abaixo são descritos os casos onde as condições são verdadeiras para cada operador:

Operador (**==**) → se x é igual a y

Operador (**!=**) → se x é diferente de y

Operador (**>**) → se x é maior que y

Operador (**<**) → se x é menor que y

Operador (**>=**) → se x é maior ou igual a y

Operador (**<=**) → se x é menor ou igual a y

Exemplo 1:

Se **x = 10** e **y = 4**, temos que, para o código:

```
if x > y:
    print("x é maior que y")
else:
    print("x não é maior que y")
```

O output seria "x é maior que y".

Exemplo 2:

```
a = int(input("Digite um número de 0 a 10"))
if a > 5:
    print("O número", a, "é maior que 5")
elif a == 5:
    print("Esse número é igual a 5")
else:
```

for_code[]

```
print("O número", a, "é menor que 5")
```

Esses operadores não estão limitados somente a números (int e float), pois também podem ser usados para comparar variáveis como strings e booleanos.

Exemplo 3:

Digamos que x = "python" e y = "hello world":

```
if x == y:
    print("As palavras são iguais")
else:
    print("As palavras são diferentes")
```

Como a palavra "python" não é a mesma coisa que "hello world", a mensagem na tela será "As palavras são diferentes". A mesma comparação pode ser feita com variáveis que carregam valores como **true** ou **false**.

6. Indentação

Percebeu que há um espaço abaixo de cada condicional? Chamamos isso de **indentação**, é uma **forma de organização necessária** para o condicional e outros tipos de estruturas como os laços de repetição (vistos mais adiante) funcionarem. Funciona como se **a estrutura fosse uma ramificação do código que executa uma série de comandos dentro do espaço indentado antes de continuar o código**. A indentação pode ser usada quantas vezes forem necessárias, só precisando manter a organização e o espaçamento. Veja o exemplo:

Exemplo 1:

```
if cafe_manha == 'café preto'
    if açúcar < x:
        adicionar_açucar()
    elif açúcar > x:
        mexer_cafe()
    beber_cafe()
```


for_code[]

No exemplo, colocamos uma condição em que o café da manhã é café preto, adicionamos o espaço de indentação e dentro dele mais condicionais (o que chamamos de aninhamento). Repare que a cada condicional damos o **espaçamento adequado para demonstrarmos a ramificação do código dentro de cada indentação**, executando os comandos dados ao programa de acordo com as condições antes de continuar o código

7. Loop

Muitas vezes você vai se deparar com situações que você precisa fazer repetir uma parte do código diversas vezes, o Python está aqui para te ajudar nisso.

Imagine que você tem dezenas de e-mails para enviar e você precisa reduzir seu trabalho automatizando a tarefa. Você pode criar uma máquina que faça o envio dos e-mails para o destinatário certo de forma repetida e com regras determinadas.

Isso é a base de um laço de repetição, agora é só imaginar que a máquina é o Python e os e-mails podem ser qualquer tipo de dado a ser processado. Os laços de repetição podem ser com intervalos definidos ou de forma contínua, podendo usar um recurso chamado iterador (explicado mais adiante)

7.1 for

O laço **for** pode ser usado, na maioria das vezes, para laços de repetição com intervalos definidos, usando um **iterador**, que será geralmente um elemento de lista ou o número da repetição do laço. Temos algumas formas comuns de usá-los, vamos a elas.

Exemplo 1:

```
for i in range(1,10):  
    print(i)
```

Nessa sintaxe, definimos o iterador como (i) e o alcance de sua repetição indo de 1 a 10. É importante lembrar que o laço se repete 9 vezes, já que nesse comando o iterador e a repetição para no penúltimo elemento. Temos como saída a impressão de 1 a 9 no terminal.

for_code[]

Exemplo 2:

```
prato = ('arroz', 'feijão', 'carne')  
  
for comida in prato:  
    print(comida)
```

Nessa sintaxe, terá uma repetição para cada elemento da lista e a cada repetição o valor do iterador será um elemento da lista, ou seja, 1º repetição - arroz, 2º repetição - feijão, e assim em diante. E como saída a esse código, teremos printado no terminal cada comida linha em linha.

Exemplo 3:

```
jantar = ('macarrão', 'molho', 'queijo ralado')  
  
for indice, comida in enumerate(jantar)  
    print(indice)  
    print(comida)
```

Nesse exemplo, além de obtermos 3 repetições e cada elemento da lista, representado por “comida”, também temos a posição do elemento, representada por “índice”, recurso que pode ser útil em algumas aplicações de análise de dados. Precisamos lembrar que o valor do índice é apenas a posição, que começa em 0 obedecendo a ordem do Python: macarrão - 0, molho - 1, queijo ralado - 2.

7.2 while

O laço de repetição **while** é usado geralmente para **repetições contínuas**, podendo até ser gerada uma repetição infinita. Na sintaxe desse laço, usamos o while acompanhado de uma condição, o laço se repetirá enquanto a condição for

for_code[]

verdadeira, e acabará imediatamente após a condição ser falsa. Vamos ao exemplo:

Exemplo 1:

```
num = 1

while num < 100

    print(num)

    num += 1
```

O Exemplo acima printa um número até 100, porém será? Nessa condição, o número só é printado até 99, pois após ser somado até 100, ele já terá saído do laço. Por isso, precisamos tomar cuidado com a lógica do programa, e para consertar esse erro, colocamos a linha “while num <= 100”. Obs: todas as variáveis dentro do laço de repetição precisam ser declaradas antes do laço ocorrer, como feito acima.

Exemplo 2:

```
num = 1

while True:

    if num == 1000

        break

    print(num)

    num += 1
```

O Exemplo acima é um claro exemplo de loop infinito, que só para quando o laço é “quebrado”. Usamos para isso o **while True** para repetirmos um laço que printa um número e soma 1 continuamente. Porém, estabelecemos uma condição que ao alcançar o número 1000 o laço é quebrado pela função **break**, que interrompe o laço à força. Como resultado do código, teremos um terminal com números de 1 a 1000 printados nele, conforme definimos no código.

8. Funções

8.1 Definindo funções

No Python, funções são sequências de comandos que executam alguma tarefa quando são chamadas pelo seu nome. Assim como as funções que são ensinadas em matemática, as funções em Python também podem receber argumentos, e realizar operações a partir deles. Para criar uma função, é preciso usar o comando **def** para definir o nome, os argumentos, e o código da função. Caso você queira que a função retorne um valor, você deve especificar com o comando **return**.

Exemplo 1:

```
def soma(a, b):  
    return a + b  
  
print(soma(5, 2))
```

No exemplo acima, apareceria 7 na tela, pois a função **soma** deve retornar a soma de dois argumentos, os quais são, no caso, 5 e 2. Note que, por retornar um valor, a função se transformou em um valor, e por isso foi necessário usar o comando **print()** para projetar o resultado na tela. No entanto, como é possível usar funções dentro de funções, podemos usar o próprio comando **print()** dentro da função **soma()**.

Exemplo 2:

```
def soma(a, b):  
    print(a + b)  
  
soma(4, 8)
```

No caso acima, não foi preciso usar a função **print()** porque ela já está incluída na função **soma()**, então foi preciso apenas invocar a função **soma()** para projetar o resultado na tela.

Outro aspecto importante das funções é que elas não precisam necessariamente de argumentos, ou seja, elas podem executar comandos apenas por serem chamadas sem nada dentro dos parênteses.

for_code[]

Exemplo 3:

```
def carro():  
    print("Vroom Vroom")  
  
carro()
```

Só de chamar a função `carro()` aparece a mensagem "Vroom Vroom" na tela, sem precisar colocar nada no argumento da função. Para ver um uso mais aplicado de funções, veja o exemplo abaixo:

Exemplo 4:

```
def contador_de_pares(lista):  
    contador = 0  
    for numero in lista:  
        if numero % 2 == 0:  
            contador += 1  
    return contador  
  
lista_de_numeros = [1, 2, 5, 6, 9, 8, 10, 12, 4]  
  
quantidade_de_pares = contador_de_pares(lista_de_numeros)  
  
print("A lista tem ", quantidade_de_pares, " números pares.")
```

A função acima pode determinar quantos números pares tem em uma lista de números qualquer. Nesse caso, o valor na tela seria de 6 números pares, mas dependendo da lista inserida no argumento, o resultado seria diferente. Imagine que você tivesse que repetir essa contagem para 100 listas. Daria muito trabalho, não é? Mas com o auxílio dessa função você poderia criar um loop para repetir a contagem em todas as listas.

8.2 Funções lambda

Existe um outro tipo de função chamado `lambda`, que é uma função sem nome e que pode ser definida em uma única linha. Para definir uma, você deve usar o seguinte modelo:

`lambda argumentos: expressão`

for_code[]

Exemplo 5:

```
soma = lambda a, b: a * b + 2
```

```
print(soma(10, 5))
```

O output seria 52. Nesse caso, a função lambda foi atribuída a uma variável, mas ela pode ser invocada diretamente.

Exemplo 6:

```
print( (lambda x, y: x**y), (5, 3) )
```

Em uma única linha de código foi definida uma função exponencial com dois argumentos, e quais argumentos seriam inseridos nela. No exemplo acima, o output seria 125, dado que 5 e 3 foram os parâmetros definidos.

As funções são utilizadas para repetir séries de comandos sem ter que escrevê-los várias vezes, então elas são muito úteis para economizar tempo e deixar seu código mais organizado.

8.3 Função Round

Existe uma função chamada round, que é utilizada para arredondar o número para um determinado número de casas decimais. A mesma é muito utilizada para cálculos financeiros, estatísticos e ciência de dados

Esta função recebe dois parâmetros: number e ndigits, em que o segundo parâmetro é opcional e determina quantas casas decimais o número aproximado terá: `round(number, [ndigits])`.

Exemplo 7:

```
import math
```

```
x = 1.223
```

```
y = math.cos(x)
```

```
print(round(y,3))
```

O output seria 0.3408269060683363 sem a função round, com a mesma se torna 0.341 com o arredondamento.

9. Bibliotecas

9.1 Matplotlib

Imagine uma situação em que você tem vários **dados e estatísticas**, e deseja mostrar essa parada da **forma mais visual possível**. Basicamente, a **biblioteca Matplotlib** é a ferramenta mágica, no mundo da programação, que **monta todos os tipos de gráficos**. E tem forma mais fácil de montar um gráfico com outra linguagem além do python? Não!! 😊

Pensa na Matplotlib como uma caneta super poderosa que desenha o que você quiser em uma folha em branco. No mundo da programação, essa "folha em branco" é uma tela virtual. Você só precisa dizer para a caneta (ou seja, o Matplotlib) como você quer que as coisas sejam desenhadas. Utilizar essa biblioteca é quase igual brincar de LEGO, iremos adicionar peça por peça.

Bora lá montar um gráfico usando essa belezinha:

Passo 1: Importar a biblioteca

Como foi comentado anteriormente sobre a biblioteca Math, aqui é a mesma coisa: Devemos "chamar" o Matplotlib para o seu código (chamar nossa heroína).

```
import matplotlib.pyplot as plt
```

Passo 2: Importar nossos dados

Bom, precisamos dos dados que queremos plotar né? Pode ser um conjunto de números, pontos ou qualquer outra coisa. No exemplo abaixo, iremos dizer que desejo relacionar dados de uma população de bactéria em função do tempo:

```
populacao = [1000, 2000, 3000, 5000, 8000]
```

```
tempo = [10, 30, 40, 50, 60]
```

Passo 3: criar o gráfico

Bora usar nossa caneta mágica 🪄

for_code[]

Iremos utilizar o comando `plt.plot()` para chamar as nossas variáveis (Repare que eu estou utilizando, por convenção, `populacao` como eixo x e `tempo` como eixo y). O comando `plt.show()` para que nosso gráfico apareça.

```
plt.plot(populacao, tempo)
```

```
plt.show()
```

Quando você executar o código, a seguinte aba irá abrir no seu computador:

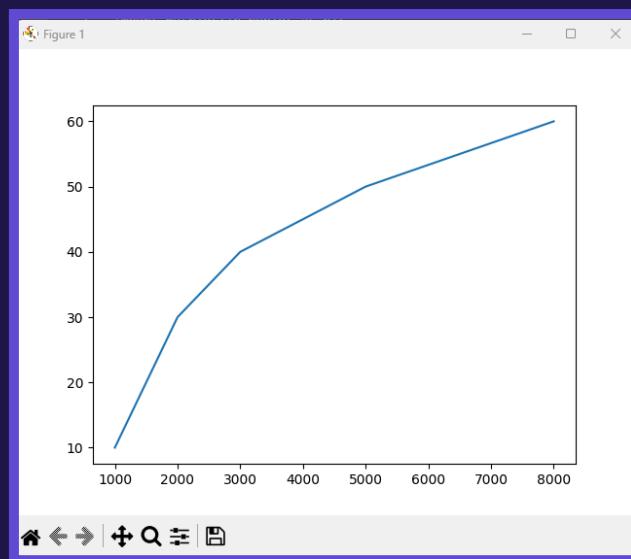


Imagem 10: gráfico plotado com matplotlib

Passo 4: Personalizar o gráfico

Se quiser dar aquele toque especial, dá pra personalizar o gráfico. Mudar as cores, adicionar título, nomear os eixos... Solta a criatividade! Todos os comandos utilizados estão explicados em forma de comentário. Não tem muito mistério!

```
# Adiciona um título ao nosso gráfico
```

```
plt.title("População de bactéria(mil) por minutos")
```

```
# Adiciona legenda ao eixo x
```

```
plt.xlabel("número de bactérias")
```

```
# Adiciona legenda ao eixo y
```

```
plt.ylabel("tempo (minutos)")
```

```
# chama o nosso gráfico
```


for_code[]

`plt.show()`

Agora que chamamos nosso gráfico depois da personalização, ele irá aparecer da seguinte forma:

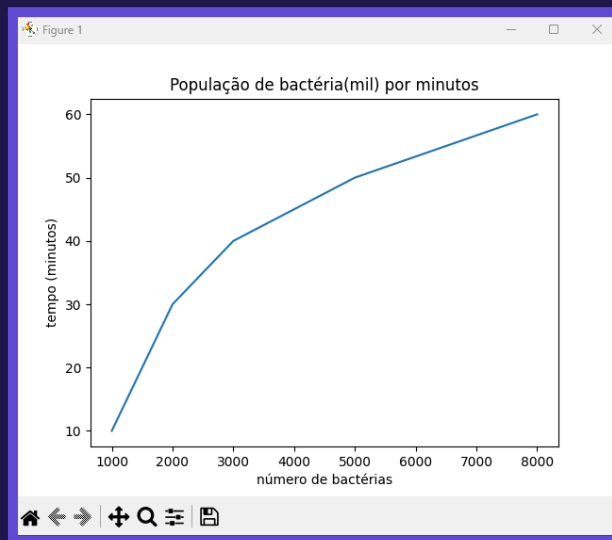


Imagem 11: gráfico personalizado matplotlib

E pronto! Você fez seu primeiro gráfico usando o Matplotlib. Agora você pode dar vida aos seus dados de um jeito visual e entender melhor o que eles estão tentando te dizer. 👍

Lembre-se que esse é só um começo, a Matplotlib tem um monte de truques e tipos de gráficos que você pode explorar conforme vai ficando mais confiante.

9.2 Numpy

NumPy é tipo aquele assistente prodígio que simplifica tudo relacionado a números. Se você é **fã de matemática** e quer fazer operações como somar, multiplicar, calcular médias e até fazer coisas malucas com matrizes, NumPy é o seu braço direito. Ah, e não precisa se preocupar com os cálculos, porque NumPy faz isso por você.

Coisas **legais** que podemos fazer utilizando a biblioteca numpy: criar uma array, cálculos com matrizes, funções matemáticas e diversas operações utilizando arrays.

Vou explicar detalhadamente cada uma dessas coisas incríveis em tópicos.

- **Criar uma array**

for_code[]

Uma array no Python é uma estrutura de dados que permite armazenar uma coleção ordenada de elementos, como números, strings ou objetos. Esses elementos são acessados através de índices, que indicam a posição de cada item na array. Em outras palavras, uma array é como uma sequência organizada de itens, onde cada item é referenciado por sua posição na sequência.

No exemplo abaixo, mostramos como montar uma array inserindo os elementos que você deseja.

```
import numpy as np

minha_lista = np.array([1, 2, 3, 4])
```

- Operações com arrays

Aqui a mágica irá acontecer! Podemos realizar diversas operações com a array que criamos, você que escolhe. Por exemplo, desejo realizar as seguintes operações:

```
minha_lista = np.array([1, 2, 3, 4])

quintuplo = minha_lista * 5

soma = minha_lista + 13
```

Repare que é preciso chamar outra variável para realizar essas operações e utilizar o comando print para que apareça na tela. O código irá nos retornar as seguinte arrays:

```
[5, 10, 15, 20]

[14, 15, 16, 17]
```

- Matrizes

Agora, iremos montar o monstro das operações matemáticas - matrizes. Relaxa que o Numpy faz isso de forma muito fácil 🤖

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

- Operações matemáticas

Não tá a fim de calcular aquelas funções difíceis? bora lá.

```
seno = np.sin(75)

exponencial = np.exp(18)
```

for_code[]

Agora, você não precisa mais passar horas calculando e manipulando números. Basta executar o código e imprimir os valores. Com o NumPy, você pode realizar essas tarefas de forma super eficiente e rápida.

9.3 Pandas

Pandas é uma biblioteca usada para trabalhar com banco de dados. Seu nome deriva do termo Panel Data, que é um conceito em inglês diretamente relacionado ao campo de estudo da econometria. A biblioteca é utilizada para limpeza e tratamento de dados, análise exploratória de dados (EDA), visualização de dados, entre outros.

A biblioteca pode ajudar na análise de dados, visualizar máximos, mínimos e a média de valores, deletar linhas não relevantes para o banco de dados.

Nos tópicos abaixo, iremos ver as principais funções da biblioteca:

- Criando uma Banco de Dados ('colunas': ['linhas'])

```
import pandas as pd

meu_banco = {
    "nomes": ["Márcia", "Ana", "Lucas"],
    "idade": [19, 20, 17]

mostrar = pd.DataFrame(meu_banco)

print(mostrar)
```

- Retornando uma linha específica

```
import pandas as pd

meu_banco = {
    "nomes": ["Márcia", "Ana", "Lucas"],
    "idade": [19, 20, 17]
}

mostrar = pd.DataFrame(meu_banco)

print(mostrar.loc[0])
```

for_code[]

- Nomeando as linhas

```
import pandas as pd

meu_banco = {

    "nomes": ["Márcia", "Ana", "Lucas"],

    "idade": [19, 20, 17]

}

mostrar = pd.DataFrame(meu_banco, index = ["cliente 1", "cliente 2", "cliente 3"])

print(mostrar)
```

- Enviando arquivos para um Banco de dados

```
import pandas as pd

#arquivo csv

banco_dados = pd.read_csv("data.csv")

print(banco_dados)
```

- Informações sobre os dados

```
#Tipos de dados

print(banco_dados.info())

#5 primeiras linhas (sem nada entre os parênteses são 5, posso enumerar
quantas linhas quero colocando o número no mesmo)

print(banco_dados.head())

#Última linha de dados

print(banco_dados.tail())

#Dados acima de algo

banco_dados[banco_dados[nome_coluna] >= valor]
```

- Plotando gráficos (utilização de Pandas e Matplotlib)

```
import pandas as pd
```

for_code[]

```
import matplotlib.pyplot as plt

banco_dados_ = pd.read_csv('data.csv')

banco_dados.plot()

plt.show()
```

10. Adorei o Python, posso virar um profissional de programação só com ele?

O Python é aquele cara que não reprova nenhuma matéria, mas também não tira a maior nota em nada. Como se fosse aquele cara que tira 8 em tudo, mas nunca 10 (analogia com a escola, 8 na faculdade é nota 😂). Porém, o Python é muito utilizado por profissionais de diversas áreas e dominá-lo é uma ótima qualidade.

Algumas das utilidades incríveis do Python incluem: Fazer Web Apps (site do Spotify, Youtube, Netflix e mais - Sites que parecem um aplicativo, simplificando), trabalhar com análise de dados e aprendizado de máquina, fazer softwares que não dependam de um desempenho absurdo - sistemas científicos ou jogos não são indicados para se trabalhar com Python, mas ele tem muitas utilidades para aplicações simples, porque se programa rapidamente sistemas grandes com ele.

11. Exercícios

1) Tendo a seguinte tabela: (dica: utilize a biblioteca Pandas)

	Nomes	Notas
Aluno 1	Maria	10
Aluno 2	Lucas	7
Aluno 3	Guilherme	3
Aluno 4	Eduarda	5

for_code[]

Aluno 5	Gabriela	7
Aluno 6	Ana	5
Aluno 7	Hugo	9

- Construa a tabela dentro do python;
- Crie um código que mostre só os alunos que tiveram nota maior ou igual a 7.

2) Crie um código que receba a informação de nota e retorne ao usuário se ele foi aprovado (maior ou igual a 7), está de recuperação (maior ou igual a 5) ou reprovado.

3) Faça um código que retorne todos os valores da lista separadamente, um abaixo do outro:

`lista = [3, 1, 12, 5, 18, 9, 10]`

4) Faça um código que receba um número e retorne, através de uma função, se o mesmo é ou não é par.

5) Faça uma função que receba um número e retorne o valor fatorial do mesmo.

6) Faça um gráfico que seja de 0 até 1 em x e plote as seguintes funções:

$$y = x^2 \text{ e } y = x$$

7) Crie um código que receba um número e retorne o seu valor ao quadrado.

8) A for_code foi contratada para fazer um projeto de cálculo, utilizando Python, para um colégio que utiliza a seguinte fórmula para calcular a média dos alunos:

$$M = (P1 + T * 2 + P2 * 4) / 7$$

Assim, você foi designado a criar um código que receba os valores para as provas (P1 e P2) e para o trabalho (T) e calcule qual foi a média do aluno, arredondando para duas casas decimais após a vírgula. Além disso, você deve avaliar se o aluno passou, está de recuperação ou foi reprovado, sabendo que a média é 5 e os alunos abaixo de 5 e até 3 estão de recuperação.

9) Uma nutricionista precisava otimizar o cálculo do IMC e, para isso, pediu a um membro da for_code para criar um código em python que fizesse o seguinte cálculo para ela:

$$IMC = \text{Peso (kg)} \div \text{Altura}^2 \text{ (m)}$$

for_code[]

Dessa forma, o código precisa receber os valores de peso e altura para, então, calcular o IMC da pessoa e retornar a sua classificação seguindo a tabela a seguir:

Valor do IMC	Classificação
< 18,5 Kg/m ²	Baixo Peso
≥ 18,5 e < 24,9 Kg/m ²	Peso Normal
≥ 25 e < 29,9 Kg/m ²	Excesso de Peso
≥ 30 e < 34,9 Kg/m ²	Obesidade Grau I
≥ 35 e < 39,9 Kg/m ²	Obesidade Grau II
≥ 40 Kg/m ²	Obesidade Mórbida

11.1 Resolução

 Resolução Documento Suporte

```
#1) a)
import pandas as pd
classe = {
    'nomes' : ["Maria", "Lucas", "Guilherme", "Eduarda", "Gabriela", "Ana", "Hugo"],
    'notas' : [10, 7, 3, 5, 7, 5, 9]
}

#Visualização da tabela
mostrar = pd.DataFrame(classe, index = ["Aluno 1", "Aluno 2", "Aluno 3", "Aluno 4", "Aluno 5", "Aluno 6", "Aluno 7"])
mostrar
```

	nomes	notas
Aluno 1	Maria	10
Aluno 2	Lucas	7
Aluno 3	Guilherme	3
Aluno 4	Eduarda	5
Aluno 5	Gabriela	7
Aluno 6	Ana	5
Aluno 7	Hugo	9

for_code[]

```
#1) b)
mostrar[mostrar['notas'] >= 7]
```

	nomes	notas
Aluno 1	Maria	10
Aluno 2	Lucas	7
Aluno 5	Gabriela	7
Aluno 7	Hugo	9

```
#2)
nota = float(input('Digite a nota >> '))

#Método de conversão
if nota >= 7.0:
    print("Aprovado")

elif nota >= 5.0:
    print("Recuperação")

else:
    print("Reprovado")

Digite a nota >> 8
Aprovado
```

```
#3)
lista = [3, 1, 12, 5, 18, 9, 10]

for c in lista:
    print(c)

3
1
12
5
18
9
10
```

```
#4)
a = int(input("Digite um número: "))

def ehpar(numero):
    return numero % 2 == 0

# Avaliando se a é par
print(ehpar(a))

Digite um número: 47
False
```


for_code[]

```
#5
b = int(input("Digite um número: "))
def fatorial(numero):
    if numero == 0 or numero == 1:
        return 1
    else:
        return numero * fatorial(numero - 1)

print(fatorial(b))
```

```
Digite um número: 5
120
```

```
#6
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0.0, 1.0, 0.001)
y1 = x**2
y2 = x

#Plotando a curva
plt.plot(x,y1, label = 'x²')
plt.plot(x,y2, label = 'x')

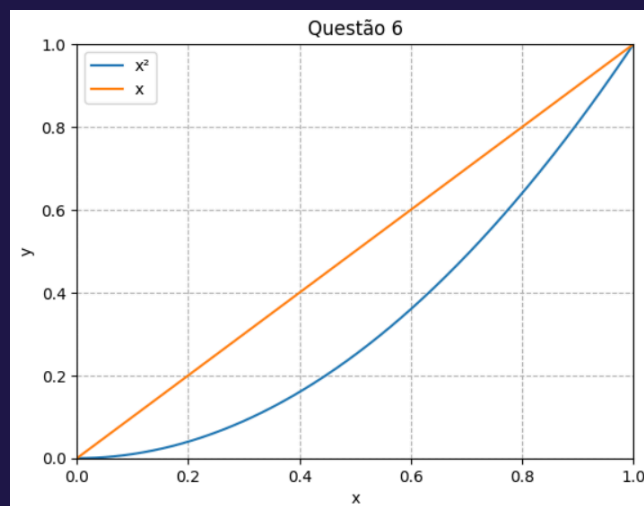
#Temos Jupyter, então não precisa de plt.show()

#Legendar
plt.legend()

#As divisões
plt.grid(ls = '--', linewidth = 0.8)

#Limites dos Eixos: funções axis
#O argumento deve ser uma lista na forma [xmin, xmax, ymin, ymax]
plt.axis([0, 1, 0, 1])

#Título e legendas dos eixos (descrição da variável)
plt.title('Questão 6')
plt.xlabel('x')
plt.ylabel('y')
```



for_code[]

```
#7
def quad(x):
    return x**2

c = int(input("Digite um número >> "))
print(quad(c))
```

```
Digite um número >> 4
16
```

```
#8
P1 = float(input("Digite a nota da 1ª Prova: "))
P2 = float(input("Digite a nota da 2ª Prova: "))
T = float(input("Digite a nota do Trabalho: "))

M = (P1 + T*2 + P2*4)/7

print(round(M,2))

if M >= 5:
    print("Você foi aprovado")
elif M >= 3:
    print("Você está de recuperação")
else:
    print("Sinto muito, mas você foi reprovado")
```

```
Digite a nota da 1ª Prova: 7
Digite a nota da 2ª Prova: 5
Digite a nota do Trabalho: 2
4.43
Você está de recuperação
```

```
#9
Peso = float(input("Digite a sua massa em quilogramas: "))
Altura = float(input("Digite quanto você tem de altura (em m): "))

IMC = Peso / Altura**2

if IMC < 18.5:
    print("Você está abaixo do peso ideal")
elif IMC >= 18.5 and IMC < 24.9:
    print("Você está na faixa de normalidade")
elif IMC >= 25 and IMC < 29.9:
    print("Você está com excesso de peso")
elif IMC >= 30 and IMC < 34.9:
    print("Você está na faixa de obesidade de primeiro grau")
elif IMC >= 35 and IMC < 39.9:
    print("Você está na faixa de obesidade de segundo grau")
else:
    print("Você está na faixa de Obesidade Mórvida")
```

```
Digite a sua massa em quilogramas: 61
Digite quanto você tem de altura (em m): 1.63
Você está na faixa de normalidade
```

OBS: Os exercícios acima foram feitos no Google Colab.

12. Conclusão

Agora você já está quase pronto para se tornar **fera** em Python. Existem diversas bibliotecas e documentações que você pode acessar para se aprofundar nessa linguagem.

Uma das melhores partes de ser um(a) **programador(a) em Python** é a incrível comunidade e os recursos disponíveis. Se você quiser mergulhar em áreas como análise de dados, aprendizado de máquina, desenvolvimento web ou automação, Python tem **ferramentas poderosas** para cada uma delas.

Se quiser expandir ainda mais seus horizontes, não deixe de conferir a **vasta gama de bibliotecas** Python além das que já mencionamos, como Scikit-Learn para aprendizado de máquina, Flask ou Django para desenvolvimento web, e muitas outras. Cada uma delas abre portas para novas habilidades e oportunidades.

E a melhor parte? A **documentação oficial** do Python é uma mina de ouro de informações. Lá você vai encontrar tutoriais, guias e exemplos detalhados que podem te ajudar a dominar as nuances da linguagem e suas bibliotecas.

Portanto, não tenha medo de explorar, experimentar e se desafiar. A jornada de se tornar um(a) ninja em Python é cheia de descobertas e conquistas. E lembre-se, mesmo os mais **experientes** continuam aprendendo **constantemente**. Então, continue praticando, construindo projetos e se divertindo com o mundo incrível da programação Python! 🐍🚀

Agradeço imensamente a quem chegou até aqui e espero que esse documento tenha **te tornado fã nº 1 de python**. A princípio esse material será utilizado como documento de apoio para o nosso Processo Seletivo e, também, difundir o conhecimento entre a Escola de Química.

Nos siga nas redes sociais:

- ◆ Instagram: [forcodeufrj](#)
- ◆ github: [for_code](#)
- ◆ LinkedIn: [for_code](#)