Introdução à programação com Python

Introdução à Programação utilizando Python

Autora
Fernanda Malheiros Assi

Revisado por
Amanda Basso de Oliveira

Janeiro de 2020

Credits
Template by Andrea Hidalgo on Overleaf
Photos by Diego PH on Unsplash

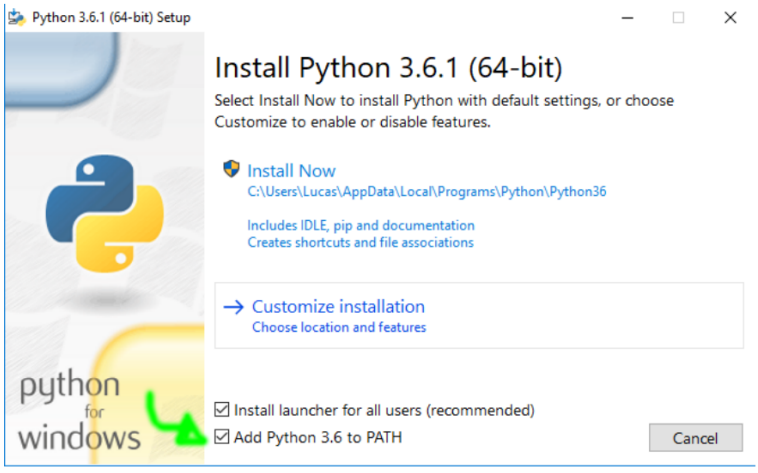
Contents

1. Preparando o Ambiente

1.1 Instalando Python no seu computador

Para instalar Python no seu computador é bem simples, basta ir no site oficial do Python na área de Downloads (`https://www.python.org/downloads/`) e clicar no botão para fazer o Download.

Em seguida, execute o instalador e uma imagem similar a essa aparecerá:



Deve ser selecionada a opção Add Python 3.6 to PATH e depois continuar a instalação até o fim.

1.2 Ambientes de Desenvolvimento

Há diversos programas para desenvolvermos códigos, alguns são mais bonitinhos, outros são mais poderosos, alguns são mais simples, outros são mais amigáveis. Dê uma olhada nesta seção e escolha o que você achar mais interessante. Somente você pode responder à pergunta "Qual o melhor ambiente de desenvolvimento para mim?".

1.2.1 ATOM

O programa ATOM é um IDE open-source que apresenta diversos pacotes para personalizar.

No site oficial do ATOM[1], você encontrará um link para a Documentação do programa. Na documentação, é possível acessar o manual[2] que mostrará passo a passo como instalar o programa (tanto para Windows como para Linux).

Caso você tenha alguma dúvida, é aconselhável entrar na seção de discusão[3]. Nessa página, você encontrará respostas para diversas dúvidas, e possivelmente, para a sua.

[1] https://atom.io/

[2] https://flight-manual.atom.io/getting-started/sections/installing-atom/#platform-windows

[3] https://discuss.atom.io/

2. A Linguagem Python

Os conteúdos dessa seção foram retirados do site `https://becode.com.br/porque-aprender-python/`

2.1 Introdução

Que a linguagem Python tem ganhado um enorme destaque nos últimos anos ninguém discute. Isso é um fato! Contudo, o que fez essa linguagem ter essa ascensão meteórica? Qual característica? Bom, se eu tivesse que creditar a ascensão da tecnologia a apenas um único fator, eu diria: simplicidade!

Dentre todas as características dessa linguagem de programação, sem sombra de dúvidas, a simplicidade é a que mais chama a atenção. Basta você tê-la instalado em seu sistema e chamá-la no console que já pode sair usando!

Python nasceu com esse objetivo: ser simples, acessível e fácil de usar. Entretanto, não podemos atribuir o sucesso da linguagem a um único fator. A tecnologia é muito mais que simplicidade. Estamos falando de uma linguagem incrível e que veio para ficar!

2.2 Um breve histórico

A implementação dessa linguagem teve início em dezembro de 1989 por Guido van Rossum. Porém, somente no início de 1991 teve a primeira versão pública (0.9.0). Já em outubro de 2000, a versão 2.0 foi anunciada e, em dezembro de 2008, a versão 3.0 que contemplava grandes mudanças internas.

Devido a sua incompatibilidade com a versão anterior, a linha de desenvolvimento 3.0 teve diversos lançamentos em paralelo. Até que finalmente, houve a comunicação oficial do fim do desenvolvimento do versão 2, em 2010, quando foi anunciado a versão 2.7.

Assim, hoje em dia, você pode utilizar qualquer uma das duas versões disponíveis, a Python 2.7 ou a versão 3. Tudo depende do que você quer fazer. Então lembre-se sempre de verificar a versão oficialmente suportada pela sua distribuição Linux, Windous ou Mac OS, já que muitas destas ainda usam a versão 2 como oficial.

2.3 Principais Características

"Tá, mas o que o Python tem de tão especial?"

Veja abaixo uma pequena lista com as características que fazem dessa tecnologia uma das mais interessantes linguagens de programação do mercado atual de desenvolvimento:

- **Interoperabilidade**: comunica-se de forma transparente com outros sistemas ou linguagens.
- **Multiplataforma**: pode ser utilizada em sistemas Linux, Mac OS ou Windows
- **Robustez**: O tópico "Aplicações da linguagem é focado nesta característica.
- **Velocidade de aprendizado**: Como o tópico a seguir já diz, é perfeita para iniciantes em programação.
- **Simplicidade**: É uma linguagem muito simples, como já foi mencionado acima.

A seguir, veremos com mais detalhes alguns dos pontos citados acima.

2.4 Perfeita para iniciantes

Aí você pode perguntar: "...mas porque ela é tão simples e fácil de usar?"

A resposta é simples: porque ela foi pensada para isso!

- Trata-se de uma linguagem de alto nível, ou seja, é mais próxima da linguagem humana do que outras linguagens de programação.
- Planejada para ser produtiva e de fácil entendimento
- Reduz a utilização de caracteres especiais (vírgula, dois pontos, ponto e vírgula, sinal de igual, chaves, etc.
- Possibilita que você determine uma sequência e monte uma lista com apenas uma linha de código.

Por ter uma sintaxe clara e direta, Python permite que você fique mais tempo desenvolvendo, focado em lógica de programação e em resolver problemas de programação. E não preocupado com especifidades da linguagem de programação.

2.4.1 Fácil de aprender?

Sim, é! Claro, sem desmerecer a profissão. Programar é algo que exige anos de prática para chegar a perfeição. Contudo, por toda a filosofia que contempla o mundo Python, a curva de aprendizado de um programador iniciante em Python é muito mais rápida do que um programador iniciante em C, por exemplo.

Em outras palavras, um programador iniciante em Python, em pouco tempo já conseguirá desenvolver o seu primeiro programa. De forma mais rápida que o programador em C, devido à complexidade envolvida no segundo caso.

2.5 Aplicações da linguagem

Sim, é uma linguagem extremamente simples, fácil de usar e aprender, mas não se engane! Apesar disso, Python também é uma linguagem extremamente robusta e utilizada nas mais diversas soluções:

- $Back-end$ de sistemas web, CRMs e ERPs
- Pesadas simulações de engenharia
- Processamento de efeitos especiais de filmes
- Soluções de análise de dados ($dataanalytics$)

- Aprendizado de máquina ($machinelearning - ML$)

2.5.1 Exemplo - O Spotfy

Bem, o Spotify usa Python em cerca de 80% dos seus serviços de back-end. Principalmente, devido ao tempo de desenvolvimento destes serviços, que deve ser o mais rápido possível para não afetar os usuários da aplicação. Outro uso da tecnologia tem relação com a análise de dados em conjunto com o Hadoop, tanto para tomadas de decisão, quanto para melhorias no produto em si.

Além disso, as opções de Radio, recomendações para você e Discover utilizam scripts Python para analisar todas as suas preferências e então criar as listas de novas opções de músicas. Machine Learning na prática! Uma vez que essa lista é criada exclusivamente de acordo com o seu gosto musical. Em outras palavras, o computador aprendeu sobre você e, com isso, adaptou o conteúdo para o seu gosto!

2.6 Foco no programador

Python foi projetada e desenvolvida com foco no programador, visando dar mais liberdade e segurança aos seus usuários. Por isso, conta com uma ampla variedade de tipos básicos, como números (ponto flutuante, complexos e inteiros longos de comprimento ilimitado), strings (ASCII e Unicode), listas, tuplas, sequências, conjuntos e dicionários.

2.7 Código aberto e a comunidade Python

Quando falamos de Python, estamos lidando com uma linguagem livre, de código aberto que permite que programadores do mundo todo compartilhem problemas, soluções e binários da linguagem, sem ter que anexar as fontes.

Além disso, a linguagem ainda conta com uma documentação extremamente rica e completa. Para encontrar a documentação, é só acessar o site oficial da linguagem[1]. Como

[1] https://www.python.org/

se não bastasse tudo isso, a linguagem ainda possui uma incrível comunidade brasileira, a Python Brasil[2].

2.8 O que vou aprender neste curso?

O objetivo deste curso é introduzir os conceitos básicos de programação para pessoas sem experiências prévias em algoritmos e programação. Para isso, pelos motivos mencionados acima, a linguagem de programação Python será utilizada.

Ao final deste curso você deverá estar familiarizado com os conceitos básicos de programação e conseguirá resolver problemas e até mesmo desafios computacionais.

Este curso também dará uma base sólida para que, caso tenha interesse, você possa continuar a aprimorar seus conhecimentos em programação e em Python em cursos de caráter intermediário e avançado.

[2]https://python.org.br/

3. Algoritmos

3.1 O que é um computador?

Antes de começar de fato a fazer algoritmos e programar, precisamos entender o que é um computador, afinal (quase) tudo o que iremos fazer neste curso envolve um computador.

Um computador é formando fundamentalmente por duas partes principais, o Software e o Hardware. O Hardware é a parte física do computador, ou seja, toda a parte eletrônica que faz com que a máquina funcione. Já o Software é a parte que nós interagimos, toda a interface, sites, programas, etc.

Estas duas partes trabalham juntas para que consigam receber e processar dados para realização de uma determinada tarefa. Ou seja, você envia uma entrada e ele tem que ser capaz de processar esta entrada e te enviar uma saída.

Mas isso ainda esta meio confuso certo? O que isto significa na prática? Vamos trabalhar com alguns exemplos.

Quando você aperta um número no seu teclado, isto é uma entrada, o seu computador processa essa informação e apresenta na tela o número que pressionou. Um outro exemplo é uma máquina fotográfica, você clica em um botão e como resultado ele te retorna uma imagem.

Um código funciona mais ou menos da mesma forma, você envia da uma entrada

e ele te retorna uma saída. O mais legal de tudo isso, é que você mesmo vai construir este código para processar este dado para que retorne a saída que você deseja.

3.2 O que é uma linguagem de programação?

Para um computador conseguir entender o que você quer que ele faça, você precisa de uma linguagem de programação. Toda linguagem de programação possui um conjunto de palavras pré definidas e um conjunto de regras sintáticas e semânticas para que seja possível especificar precisamente como o computador deve lidar com os dados.

Parece familiar? Uma língua, como por exemplo a língua portuguesa, também possui um conjunto de palavras pré-definidas, e um conjunto de regras gramaticais para que um aglomerado de palavras façam sentido para quem as ouve.

Para efeito de comparação vamos pensar que nós somos o computador e que uma determinada língua é uma linguagem de programação. Um falante da língua portuguesa consegue processar uma frase em português, entender, e responder. No entanto ele não consegue processar uma frase em japonês. Um computador é da mesma forma, ele consegue processar uma determinada linguagem de programação e emitir uma saída.

3.3 O que é um algoritmo?

Por definição, em ciência da computação, um algoritmo é uma sequência finitas de ações executáveis que visam obter uma solução para um determinado tipo de problema. Algoritmos são procedimentos precisos, não ambíguos, mecânicos, eficientes e coretos.

Um programa de computador é essencialmente um algoritmo que diz ao computador os passos específicos que ele deve seguir e qual a é a ordem que ele deve realiza-los. Por isso é muito importante que eles sejam finitos e não ambíguos, pois um para uma dada entrada, um programa sempre deve retornar a mesma saída em um tempo finito.

3.3.1 Exemplo

Agora que você entendeu o que é um algoritmo e para que ele serve vamos trabalhar com um exemplo. Vamos construir um algoritmo para fazer um omelete. Para isso, vamos listar as ações por ordem:

1. Pegar os ovos
2. Pegar a frigideira
3. Pegar óleo ou manteiga
4. Acender o fogo
5. Fritar os ovos

Embora esta lista possa até parecer correta, ainda falta diversas instruções para que este algoritmo esteja correto. Por exemplo, o que eu devo fazer para fritar os ovos? É necessário colocar os ovos na frigideira? Embora parece bem obvio para nós humanos, esta não é uma pergunta tão obvia para um computador. Além disso, qual que eu devo pegar, o óleo ou a manteiga? Perceba que essa sentença é ambígua e confunde o computador.

Vamos tentar novamente, mas agora com instruções mais claras e mais específicas, sem ambiguidade.

1. Pegar os ovos
2. Pegar a frigideira
3. Pegar manteiga
4. Pegar uma colher de pau
5. Acender o fogo
6. Colocar o óleo na frigideira
7. Quebrar os ovos
8. Por o conteúdo dos ovos na frigideira
9. Mexer o ovos com a colher de pau por 2 minutos
10. Apagar o fogo
11. Pegar um prato
12. Colocar o conteúdo da frigideira dentro do prato

O algoritmo acima ainda pode ser mais específico, por exemplo, onde que eu pego a

frigideira, a manteiga, colher de pau, etc? Se eu pego a frigideira no armário eu ainda preciso abrir o armário antes, estender meu braço abrir a palma da minha mão, etc. No entanto, para o entendimento de como funciona um algoritmo, este nível de precisão já está de bom tamanho.

Vale ressaltar que entender como faz um algoritmo é de extrema importância se você quer aprender bem a programar. Por isto, este capítulo possui diversos exercícios para fazer algoritmos e é muito importante que vocês façam todos eles. Além disso, seria legal se, no dia-a-dia, ao realizar uma tarefa vocês já começassem a pensar algoritmicamente em como vocês as estão realizando.

3.4 Como Python executa um programa?

Até aqui, estamos tratando Python como uma linguagem de programação, mas após implementado, ele também é um pacote de software chamado de *interpretador*. Basicamente, um interpretador é um tipo de programa que executa outros programas. Quando você escreve programas em Python, o interpretador lê linha por linha executando as instruções que ele contém.

Ou seja, você escreve um programa em Python e quando você termina, você executa ele para obter uma determinada saída. O interpretador vai ler linha por linha do seu programa, executando as ações especificadas para então obter uma determinada saída.

3.5 Exercícios

1. Escreva com as suas próprias palavras o que é um algoritmo.
2. Escreva um algoritmo para ir de onde você estiver até a sua casa. (se você estiver em casa imagine que você está no último lugar que você foi).
3. Escreva um algoritmo para calcular o troco de uma compra
4. Escreva um algoritmo para somar dois números
5. Escreva um algoritmo para beber um copo de água
6. Escreva um algoritmo para lavar a louça.

4. Expressões Numéricas

4.1 Operadores matemáticos

A linguagem Python possui operadores, que são símbolos especiais utilizados para representar operações de cálculos, ou operações lógicas. Neste momento focaremos somente nos operadores matemáticos ou operadores aritméticos.

Estes são:

Lista de Operadores			
Nome	Símbolo	Exemplo	Saída
Soma	+	10 + 3	13
Subtração	-	10 - 3	7
Multiplicação	*	10*3	30
Divisão	/	10 / 3	3.3333333333333335
Divisão inteira	//	10 // 3	3
Módulo	%	10 % 3	1
Exponenciação	**	10**3	1000000

4.1.1 Divisão *vs* Divisão inteira

Qual é a diferença entre a divisão e a divisão inteira?

Enquanto a divisão irá te mostrar o resultado completo, com todas as casas decimais, a divisão inteira irá resultar apenas na parte inteira, desconsiderando a parte decimal. É importante ressaltar que ela não arredonda o número, por exemplo 10/6 é igual a 1 e não 2.

Outra coisa importante para tomar nota é que quando o resultado da divisão é um número exato, a divisão normal coloca uma casa decimal (o zero), enquanto a divisão inteira não coloca nada. Por exemplo: Na divisão normal 10/5 resulta em 2.0 já na divisão inteira 10//5 resulta apenas em 2.

4.1.2 O que é o Módulo?

O módulo não é nada mais nada menos do que o resto de uma divisão. No exemplo dado na tabela o módulo de 10%3 resulta em 1 pois o resto da divisão de 10 por 3 é 1.

4.1.3 Radiação

Mas e se eu quiser tirar a raíz quadrada de um número, por exemplo, como que eu faço?

Bom, vale lembrar que a raíz de qualquer número nada mais é do que uma forma diferente de escrever a exponenciação. Por exemplo, a raiz quadrada de 4 ($\sqrt{4}$) pode ser reescrita como 4 elevado a um meio ($4^{1/2}$).

No entanto, para raízes quadradas o Python possui uma maneira, basta usar a função *sqrt* da biblioteca *math* da seguinte maneira:

```
1 import math
2 math.sqrt(4)
```

Na primeira linha do código estamos importando a biblioteca *math* para que assim possamos usar a função *sqrt*. As importações de todas as bibliotecas que serão usadas no código são normalmente feiras nas primeiras linhas do programa.

4.2 Expressões numéricas

Agora que você sabe os operadores básicos do Python está na hora de aprender a combina-los para formar uma expressão numérica.

4.2.1 Ordem dos Operadores

Assim como na matemática, em Python também existe uma ordem dos operadores a ser seguida, isto serve para não haver ambiguidade nas expressões. O acrônimo **PEMDAS** serve para nos ajudar a lembrar da ordem:

- **P**arênteses
- **E**xponenciação
- **M**ultiplicação e **D**ivisão
- **A**dição e **S**ubtração

4.3 Variáveis

Uma variável é um nome que você da a uma posição de memória do seu computador que contém uma expressão ou um valor. Para ficar mais fácil, imagine uma variável como se fosse uma caixinha que nela você guarda alguma coisa.

É importante que no seu código as variáveis tenham um nome significativo, para que você possa entender posteriormente qual informação que ela armazena. As variáveis podem ter o tamanho que você desejar e podem conter letras e números. Elas também podem começar com letras maiúsculas, porém por convenção letras maiúsculas no início do nome de uma variável não é muito utilizado.

As variáveis não podem:

- Começar com um número
- Palavras reservadas do Python
- Caracteres especiais com exceção do underline _
- Caracteres que não estejam no alfabeto Inglês

4.3.1 Declarando uma Variável

Ok, agora que eu sei o que é uma variável, como que eu faço para utilizar ela no meu código?

Para armazenar um valor em uma variável basta escrever o nome da variável, o sinal de igual e o que ela recebe.

Exemplo:

```
1 nota = 7.2
```

Neste exemplo foi criada uma nova variável, nota, e nela foi colocado o valor 7.2. Imagine como se eu pegasse uma caixinha e desse o nome a ela de "nota" e dentro dela colocasse o valor 7.2

4.3.2 Variável recendo variável

É possível que uma variável receba como valor outra variável?

A resposta para essa pergunta é sim. Nós fazemos isso da mesma forma que fizemos para atribuir um valor, mas é importante que a variável que esta sendo atribuída possua alguma coisa dentro dela.

Por exemplo:

```
1 numero1 = 8.4
2 numero2 = 9.0
3 soma = numero1 + numero2
```

Neste exemplo, colocamos 8.4 dentro da variável "numero1" e 9.0 dentro da variável "numero2", em seguidas criamos uma nova variável, "soma" e nela colocamos a soma de "numero1" e "numero2". Desta forma, se "abrirmos a caixinha" soma, ou seja, se olharmos o que tem dentro da variável "soma" encontraremos o número 17.4.

4.3.3 Tipo de uma variável

Em Python, cada variável possui um tipo. Os principais tipos são:

- int: números inteiros

- float: números reais
- str: strings (cap 6)
- list: listas (cap 5)

Para saber qual é o tipo de uma determinada variável basta escrever $type(nome\_da\_variavel)$. Por exemplo:

```
1 numero = 7
2 type(numero)
```

4.4 Entrada e Saída de dados

Neste ponto você já sabe como atribuir um valor a uma determinada variável no seu programa, mas na vida real o programador nem sempre possui os valores para colocar nas variáveis, quem estiver executando o programa é quem sabe.

Desta forma, é preciso fazer o programa de alguma forma que na hora que ele está sendo executado, ele "pergunte" quais são os valores das variáveis para então atribuí-las em tempo de execução.

Muitas vezes também é preciso mostrar na tela o alguma informação, seja o resultado da execução do programa ou alguma comunicação mais simples para quem está executando o programa.

4.4.1 Input

O input é uma função de entrada de dados, ela irá receber algum tipo de dado do teclado em tempo de execução.

Por natureza, o tipo que ela espera receber é sempre uma string, se você espera receber qualquer outro tipo de dado que não seja uma string, é necessário informar isto a ela.

Além disso esta função permite que você escreva para o usuário alguma mensagem, normalmente relativa ao que você quer que ele informe.

Vamos a alguns exemplos para ficar mais fácil de entender:

Exemplo 1:

Código:

```
1 nome = input("Informe seu nome: ")
```

Saída:

Neste caso, o programa emite uma mensagem para o usuário que está executando o programa. O programa espera que o usuário digite alguma coisa e pressione o "enter". Assim que o "enter" for pressionado, o que o foi digitado será atribuído a variável "nome".

Exemplo 2:

Mas e se o valor que eu estiver esperando for um inteiro, por exemplo? Neste caso é necessário informar o tipo que você está esperando antes da função *input*().

Código:

```
1 nota = float(input("Informe a nota: "))
```

Saída:

Neste caso, assim como no caso anterior, o programa emite uma mensagem ao usuário, no entanto, ele espera que o valor a ser informado seja um *float*. Caso isto não aconteça o programa gera um erro, exceto se for informado um inteiro, pois neste caso, o próprio programa converte para float.

4.4.2 Print

A saída de dados, além de deixar o programa mais genérico, muitas vezes também é responsável por informar ao usuário informações relevantes do programa em questão. Para que seja possível imprimir alguma coisa na tela, o mais comum é que o programador utilize a função print (embora possua outros meios que não serão abordados aqui).

Exite diversas maneiras de imprimir texto e variáveis na tela, aqui eu vou mostrar alguns exemplos daqueles que acho que são mais interessantes no dia-a-dia. O link para acesso completo as formas de saída de dados é: `https://docs.python.org/3/tutorial/inputoutput.html`

Exemplo 1:

Código:

```
1 print("{} possui {} anos." .format("Fernanda", 20))
```

Saída:

```
Fernanda possui 20 anos.
```

Neste caso, no lugar das chaves será colocado o que está entre parenteses no .format em sua respectiva ordem, que é separado por vírgulas. Na primeira chaves é colocado *Fernanda* e na segunda chaves é colocado o número 20.

Note que as aspas dentro do .format para o nome Fernanda é importante pois diz ao programa que é uma *string*. Se as aspas não estivessem lá, o interpretador assumiria que "Fernanda" é o nome de uma variável e procuraria para ver qual valor ela armazena. Como em "Fernanda" não foi colocado nada, já que neste programa ela não é uma variável, o programa retorna um erro:

Código:

```
1 print("{} possui {} anos." .format(Fernanda, 20))
```

Saída:

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-b1f212a5d194> in <module>
----> 1 print("{} possui {} anos." .format(Fernanda, 20))

NameError: name 'Fernanda' is not defined
```

Exemplo 2:

Código:

```
1 nome = "Fernanda"
2 idade = 20
3 print(nome, "possui", idade, "anos")
```

Saída:

```
Fernanda possui 20 anos.
```

Neste exemplo, as variáveis, que são tudo que não estão entre aspas, são substituídas pelo seu valor. Ou seja, *nome* será substituído por *Fernanda* e *idade* será substituído por 20. É importante notar que quando uma variável é colocada, um espaço é automaticamente inserido no texto.

4.5 Comentários

Comentários são linhas de código que são ignoradas pelo interpretador. Elas servem para que o programador consiga explicar o que o trecho de código faz, para que, desta forma se outras pessoas(ou até ele mesmo no futuro) precisarem usar o código, fique mais fácil de entender.

Comentários são uma boa prática de programação e são muito utilizados. Mas como eu faço para colocar um comentário no meu código?

Em Python exite duas formas: Se você quiser somente comentar na linha, utiliza-se o *hashtag*, que fará com que o restante da linha vire comentário. Já se você quer comentar um trecho maior utiliza-se três aspas simples no começo e mais três aspas simples no final do comentário

Comentário por Linha

```
1 # Tudo o que estiver nesta linha sera comentario para python
```

Comentário em blocos

```
1 '''
2 Tudo o que estiver dentro deste
3 bloco sera comentario para Python
4 '''
```

4.6 Exercícios

1. Calcule o resto da divisão de 10 por 3.
2. Calcule a tabuada do 13.
3. Davinir não gosta de ir às aulas. Mas ele é obrigado a comparecer a pelo menos 75% delas. Ele quer saber quantas aulas pode faltar, sabendo que tem duas aulas por semana, durante quatro meses. Ajude o Davinir!

   obs: Para este problema, considere o mês tendo 4 semanas.
4. Calcule a área de um círculo de raio $r = 2$.

   Lembrete: a área de um círculo de raio r é: $A_0 = \pi r^2$
5. Quantos segundos há em 3 horas, 23 minutos e 17 segundos?
6. Se você correr 65 quilômetros em 3 horas, 23 minutos e 17 segundos, qual é a sua velocidade média em m/s?
7. Resolva essa expressão:

$$\frac{100 - 413 * (20 - 5 * 4)}{5} \tag{4.1}$$

8. Enivaldo quer ligar três capacitores, de valores:
   - $C_1 = 10\mu F$
   - $C_2 = 22\mu F$
   - $C_3 = 6.8\mu F$

   Se ele ligar os três em paralelo, a capacitância resultante é a soma:

$$C_p = C_1 + C_2 + C_3 \tag{4.2}$$

   Se ele ligar os três em série, a capacitância resultante é:

$$C_s = \frac{1}{\frac{1}{C_1} + \frac{1}{C_2} + \frac{1}{C_3}} \tag{4.3}$$

   Qual é o valor resultante em cada um desses casos?
9. Refaça o exercício anterior, porém para quaisquer valores de capacitores.
10. Leia um nome pelo teclado e imprima "Olá, <nome lido>!"
11. Leia o nome do aluno, e a nota das três provas dele. Em seguida, calcule sua média (média aritmética simples)

12. Leia o nome de uma pessoa, a sua idade, altura, sexo, peso e se é fumante ou não. Em seguida, imprima uma fixa desta pessoa com essas informações.

5. Listas

5.1 O que é uma lista?

Uma lista em Python é uma maneira de organizar os dados para facilitar o seu acesso. Em uma única lista é possível armazenar inteiros, floats, stings e até mesmo outras listas. Ela é nada mais nada menos do que uma sequência de dados onde nela você pode adicionar novos valores, remover e acessar (consultar) um valor em uma determinada posição.

5.2 Declarando uma lista

Para declarar uma lista, basta escolher seu nome, colocar um sinal de atribuição (=) e abrir e fechar colchetes.

```
1 listaVazia = []
```

Neste exemplo a lista criada foi uma lista vazia, ou seja, a lista está criada na memória, porém não há nada dentro dela ainda. No entanto, é possível criar uma lista já adicionando valores dentro dela.

Código

```
1 lista = ['Fernanda', 20, 'F', 56, 1.58]
```

5.2.1 Lendo uma lista do teclado

Assim como variáveis, em uma lista também é possível ler os valores direto do teclado, com uma pequena diferença. Para listas usamos a função *split*() que irá separar os elementos. Por regra, ela separa quando o usuário pressiona a tecla *espao*, no entanto é possível modificar isso, como veremos em seguida.

Além disso, quando os valores são lidos do teclado, por regra, o Python assume que os valores são strings, então mesmo se um número for inserido, ele será considerado como um texto. Desta forma não é possível realizar operações matemáticas com ele. (Veremos mais pra frente como mudar isso)

Exemplo 1:

Código

```
1 lista = input().split()
2 print(lista)
```

Para a entrada: *Fernanda 20 F 56 1.58* a saída é:

```
Fernanda 20 F 56 1.58
['Fernanda', '20', 'F', '56', '1.58']
```

Mas e se eu quiser escrever o nome completo de uma pessoa, por exemplo, Fernanda Malheiros Assi? Da forma como foi feito, criar um novo item da lista cada vez que o usuário digitar o botão de espaço se torna inviável.

A função *split*() permite que seja usado outros caracteres como forma de separação de um item e outro da lista. Para isso, basta escrever o conjunto de caracteres que você deseja usar como separador entre aspas dentro do parênteses da função *split*() como é mostrado no exemplo a seguir.

Exemplo 2:

Código

```
1 lista = input().split(", ")
```

```
print(lista)
```

Saída

```
Fernanda Malheiros Assi, 20, F, 56, 1.58
['Fernanda Malheiros Assi', '20', 'F', '56', '1.58']
```

Neste exemplo, o interpretador sabe que o usuário está inserindo um novo elemento na lista quando ele encontra o caractere vírgula seguido de um espaço. Desta forma é possível inserir o nome completo de uma pessoa sem que crie-se novos elementos da lista.

5.3 Índices

Muitas vezes é preciso conseguir acessar um elemento em uma determinada posição da lista, seja para modifica-lo, removê-lo, fazer alguma operação com ele ou somente para consultá-lo.

Para isso existe os índices, são eles os responsáveis por tornar possível esse acesso. Os índices começam do 0 e vão incrementando de um em um até o último elemento da lista. A seguir, serão mostrados alguns exemplos de como isso pode ser feito.

Exemplos:

```
lista = [1,2,3,4,5]
print(lista[0])
print(lista[3])
```

Saída

```
1
4
```

Neste caso, estamos acessando elemento por elemento, no entanto, é possível acessar mais de um elemento por vez:

```
lista = [1,2,3,4,5]
print(lista[0:2])
print(lista[2:5])
```

Saída

```
[1, 2]
[3, 4, 5]
```

Essa forma de acessar elementos da lista se chama fatiamento. Ela pega o primeiro índice e vai até o último índice, porém não o considera. Ainda é possível aprofundar mais no fatiamento, adicionando um passo:

```
1 lista = [1,2,3,4,5]
2 print(lista[0:2:1])
3 print(lista[2:5:2])
```

Saída

```
[1, 2]
[3, 5]
```

Como podemos observar no exemplo acima, $lista[0:2:1]$ pega os elementos do índice 0 até os de índice 2 (1,2) e vai de um em um, dando o resultado de [1,2]. Já o segundo exemplo $lista[2:5:2]$ pega os elementos do índice 2 até os de índice 5 (3,4,5) e vai de dois em dois, dando o resultado de [3,5].

5.4 Operações em uma lista

Em Python, existem diversas funções com listas que facilitam a vida do programador (e muito!). A seguir vou mostrar as principais.

- **len(L)**: Retorna o tamanho de uma lista:

```
1 lista = [1,2,3,4,5]
2 print(len(lista))
```

Saída: 5

- **append(L)**: Adiciona um item ao final da lista:

```
lista = [1,2,3,4,5]
lista.append(6)
print(lista)
```

Saída: [1, 2, 3, 4, 5, 6]

- **extend(L)**: Estende a lista adicionando ao fim dela a lista passada como parâmetro:

```
lista = [1,2,3,4,5]
lista2 = [6,7,8]
lista.extend(lista2)
print(lista)
```

Saída: [1, 2, 3, 4, 5, 6, 7, 8]

- **insert(i, x)**: Insere o valor x na posição i da lista:

```
lista = [1,2,4,5]
lista.insert(2, 3)
print(lista)
```

Saída: [1, 2, 3, 4, 5]

- **count(x)**: Retorna o número de vezes que o valor x aparece na lista:

```
lista = [1,2,4,2,5,2,2]
print(lista.count(2))
```

Saída: 4

- **pop(i)**: Remove o item na posição dada e o retorna. Caso nenhuma posição seja especificada (a.pop()), remove e retorna o último item na lista

```
# Com parametro
lista = [1,2,3,3,4,5]
lista.pop(2)
print(lista)
```

Saída: [1, 2, 3, 4, 5]

```
# Sem parametro
lista2 = [1,2,3,4,5,5]
lista2.pop()
print(lista2)
```

Saída: [1, 2, 3, 4, 5]

- **remove(x)**: Remove o primeiro item da lista cujo valor é x. Caso não existe nenhum valor x na lista, o programa retorna um erro:

```
lista = [1,2,4,3,4,5]
lista.remove(4)
print(lista)
```

Saída: [1, 2, 3, 4, 5]

- **sort()**: Ordena os itens na lista sem gerar uma nova lista:

```
lista = [3,2,5,1,4]
lista.sort()
print(lista)
```

Saída: [1, 2, 3, 4, 5]

- **reverse()**: Inverte os itens da lista sem gerar uma nova lista:

```
lista = [1,2,3,4,5]
lista.reverse()
print(lista)
```

Saída: [5, 4, 3, 2, 1]

- **in** Mostra se determinado valor x pertence a lista:

```
lista = [1,2,3,4,5]
print(3 in lista)
```

Saída: True

5.5 Exercícios

1. Crie uma lista com o nome de 3 pessoas.
2. Crie três listas, uma lista de cada coisa a seguir:
   - frutas
   - docinhos de festas (não se esqueça dos brigadeiros)
   - ingredientes de feijoada

Lembre-se de salvá-las em alguma variável!

   (a) Agora crie uma lista que contenha essas três listas (uma lista de lista, vamos chamar de *listona*).

   (b) Você consegue acessar o elemento *brigadeiro*?

   (c) Adicione mais *brigadeiros* à segunda lista de listona.

3. Dado uma lista de números (deve ser lida do teclado), ordene-a e em seguida inverta a ordem desta lista.
4. Leia uma frase e retorne a primeira e a última palavra
5. Faça um programa que peça o nome completo de uma pessoa e retorne o primeiro nome.
6. Dada a seguinte lista: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] imprima somente os números pares dessa lista.

6. Strings

6.1 O que é uma string?

Strings são tipos de dados que armazenam uma cadeia de caracteres, sendo que eles podem ser numéricos, letras do alfabeto ou caracteres especiais. Em python, uma string é representada estando dentro de aspas simples ’ ou duplas "".

6.2 Operações com strings

Na programação é comum a manipulação de strings, visto que uma boa parte das informações que o ser humano trabalha são textos. Por isso, saber manipular strings é um tópico muito importante na computação. Python é uma linguagem que possui diversas funções já prontas para esta tarefa, o que torna muito mais fácil a vida do programador.

A seguir, eu vou mostrar as mais importantes:

- **Concatenação**: Concatena (junta) duas strings:

```
1 string = "bolo de "
2 palavra = "banana"
3 frase = string + palavra
4 print(frase)
```

Saída: bolo de banana

- **Repetição**: Repete uma string um número x de vezes:

```
palavra = "banana "
repete = palavra * 4
print(repete)
```

Saída: banana banana banana banana

- **Indexação**: Assim como nas listas, também é possível saber o que possui em determinada posição da string:

```
palavra = "banana"
print(palavra[1])
```

Saída: a

- **Fatiamento**: E também é possível realizar o fatiamento da mesma forma que nas listas:

```
palavra = "banana"
print(palavra[1:3])
```

Saída: an

- **Substituição**: Substitui o que está entre colchetes pelo o que está dentro do parênteses em .format():

```
print("bana{}na".format(10))
```

Saída: bana10na

- **Verificação**: Verifica se determinada string está contida em uma outra string:

```
palavra = "banana"
print("ban" in palavra)
```

Saída: True

- **upper()**: Deixa todos os caracteres em letras maiúsculas:

```
palavra = "banana"
print(palavra.upper())
```

Saída: BANANA

- **lower**: Deixa todos os caracteres em letras minúsculas:

```
palavra = "BANANA"
```

```
print(palavra.lower())
```

Saída: banana

- **capitalize()**: Deixa a primeira letra da string maiúscula:

```
palavra = "banana"
print(palavra.capitalize())
```

Saída: Banana

- **count(x)**: Conta quantas vezes determinada sequência de caracteres x aparece em uma string:

```
palavra = "banana"
print(palavra.count('a'))
```

Saída: 3

- **replace(x,y)**: Substitui a sequência de caracteres x pela sequência de caracteres y em uma string:

```
palavra = "banana"
print(palavra.replace('a', '-'))
```

Saída: b-n-n-

- **find(x)**: Encontra a posição que o primeiro caractere da string x aparece na string que está sendo usada como base:

```
palavra = "banana"
print(palavra.find('na'))
```

Saída: 2

- **len(x)**: Retorna o tamanho da string x:

```
palavra = "banana"
print(len(palavra))
```

Saída: 6

6.3 Exercícios

1. Dada a frase "Python é muito legal", use o fatiamento para dar nome as variáveis contendo cada palavra.
2. Qual o tamanho dessa frase? E qual o tamanho de cada palavra? Use o artifício da programação para saber se você estava certo.
3. Leia duas strings do teclado e veja se uma contém a outra.
4. Pegue o seu nome e transforme todas as letras em maiúsculas, em seguida transforme todas as letras em minúsculas e por fim, deixe só a primeira letra como maiúscula.
5. Leia uma string do teclado e faça ela se repetir 10 vezes.

7. Condicionais

7.1 Para que serve?

Estruturas condicionais são ferramentas muito úteis na computação, elas permitem que um trecho de código seja executado somente se satisfazer uma dada condição.

Vamos pensar no seguinte problema, um aluno é considerado aprovado se a nota dele é igual ou superior a 6, e reprovado caso contrário. Você quer que, dado a nota de um aluno, o programa imprima se ele foi aprovado ou reprovado. Com seus conhecimentos até o momento, fazer este tipo de programa ainda não é possível.

Antes de ver exatamente a estrutura e como funciona exatamente uma condicional, vamos estudar um pouco sobre os operadores lógicos e a lógica booleana para conseguirmos ter uma base melhor.

7.2 Operadores Relacionais

No capítulo 4 vimos o que são operadores aritméticos, que fazem uma operação entre dois valores e retornam um outro valor que também é um valor aritmético. Os operadores relacionais seguem a mesma lógica, mas ao invés de retornar um valor numérico, ele retorna um valor booleano (verdadeiro ou falso).

A tabela abaixo mostra quais são os operadores relacionais:

Lista de Operadores			
Nome	Símbolo	Exemplo	Saída
Igual	==	10 == 3	False
Diferente	!=	10 != 3	True
Menor	<	10 < 3	False
Maior	>	10 > = 3	True
Menor igual	<=	10 <= 3	False
Maior igual	>=	10 >= 3	True

7.2.1 Sinal de = *vs* ==

Enquanto na matemática o sinal de igual (=) representa uma igualdade entre dois valores, em Python isso não é verdadeiro. O sinal de igual repetido somente uma vez significa atribuição, ou seja, que uma dada variável está recebendo alguma coisa. Já o sinal de igual repetido duas vezes signidica igualdade, está comparando para ver se um valor é igual a outro ou não.

7.3 Lógica booleana

Existem também os operadores lógicos, eles comparam dois valores booleanos e retornam também um valor booleano. Para cada operador lógico existe uma tabela verdade, e o conjunto desses operadores lógicos formam o que chamamos de lógica booleana.

A seguir, será mostrado os quatro operadores lógicos com suas respectivas tabelas:

- **and (e)**: Dados dois valores booleanos p e q, o operador *and* resulta em True (verdadeiro) somente se p e q forem ambos verdadeiros (True), e resulta em False (falso) caso contrário:

p	q	p *and* q
True	True	True
True	False	False
False	True	False
False	False	False

- **or (ou)**: Dados dois valores booleanos p e q, o operador *or* resulta em False (falso) somente se p e q forem ambos falsos (False) e resulta em True (verdadeiro) caso contrário:

p	q	p *or* q
True	True	True
True	False	True
False	True	True
False	False	False

- **not (não)**: O operador *not* muda o valor do argumento, ou seja, se uma condição era True (verdadeira), ela vira False (falsa) e vice-versa:

p	*not* p
True	False
False	True

7.4 Condicional Simples - If (se)

Como já vimos, quando exite uma instrução no código que precisa ser executada apenas se satisfizer uma condição, é preciso usar uma condicional.

Em Python, a sintaxe do if é a seguinte:

```
1 if condicao:
2     consequencia
```

É de extrema importância prestar atenção na indentação do código. Após a condição é necessário colocar dois pontos para dar início ao bloco. Em baixo, é necessário dar "tab"

em todas as linhas da consequência para indicar que as instruções fazem parte do mesmo bloco de código.

Exemplo 1

Vamos pegar o exemplo dado no início do capítulo: um aluno é considerado aprovado se a nota dele é igual ou superior a 6, e reprovado caso contrário. Você quer que, dado a nota de um aluno, o programa imprima se ele foi aprovado ou reprovado.

Em Python, o código para este programa fica da seguinte maneira:

```
1 nota = 7
2
3 if nota >= 6:
4     print("Aprovado")
5 if nota <  6:
6     print("Reprovado")
```

Saída: Aprovado

Exemplo 2

Agora vamos fazer um programa que imprime se for segunda-feira imprime que é dia de faxina, se for sexta imprime que é dia de mercado e que se for qualquer outro dia da semana imprime que é dia normal:

```
1 dia_semana = 'quarta-feira'
2
3 if dia_semana == 'segunda-feira':
4     print("Dia de faxina")
5 if dia_semana == 'sexta-feira':
6     print("Dia de mercado")
7 if dia_semana == 'terca-feira':
8     print("Dia normal")
9 if dia_semana == 'quarta-feira':
10    print("Dia normal")
```

```
11 if dia_semana == 'quinta-feira':
12     print("Dia normal")
13 if dia_semana == 'sabado':
14     print("Dia normal")
15 if dia_semana == 'domingo':
16     print("Dia normal")
```

Saída: Dia normal

Perceba que para este exemplo é cansativo e inviável escrever todos os dias da semana que é dia normal, já que eles são a maioria. Existe uma maneira de simplificar esse código que veremos a seguir.

7.5 Condicional Composta - else (senão)

É formada por um comando quando a condição é cumprida (if) e um outro comando para quando a condição não for cumprida (else). O else só funciona quando tem um if previamente e segue a mesma regra de indentação vista anteriormente.

Em Python essa sintaxe tem o seguinte formato:

```
1 if condicao:
2     acontece uma consequencia
3 else:
4     caso contrario acontece essa consequencia
```

Exemplo:

Vamos refazer o exemplo 2 da seção anterior: vamos fazer um programa que imprime se for segunda-feira imprime que é dia de faxina, se for sexta imprime que é dia de mercado e que se for qualquer outro dia da semana imprime que é dia normal:

```
1 dia_semana = 'quarta-feira'
2
3 if dia_semana == 'segunda-feira':
4     print("Dia de faxina")
5 if dia_semana == 'sexta-feira':
```

```
    print("Dia de mercado")
else:
    print("Dia normal")
```

Saída: Dia normal

7.6 Condicional Aninhada - elif e else

A condicional aninhada é usada em situações onde é preciso realizar condições de teste sucessivas, em que uma ação será executada somente se um conjunto anterior de ações seja satisfeito.

No exemplo anterior, usamos dois $if's$, no entanto, embora funcione, não é a forma mais eficiente de resolver a questão. O problema com essa forma de resolução é que ele vai olhar para todos os $if's$, mesmo se o primeiro já satisfazer a condição. Usando condicional aninhada, assim que uma condição for satisfeita, o programa pula as outras condições, tornando o código mais eficiente.

A sintaxe da condicional aninhada é feita da seguinte maneira:

```
if condicao:
    acontece consequencia
elif condicao:
    acontece consequencia
else:
    acontece consequencia
```

Exemplo 1:

Um aluno é considerado aprovado, se a nota dele for maior ou igual a 6, é considerado de recuperação se a nota for inferior a 6 mas superior a 5, e é considerado reprovado se sua nota for menor que 5. Escreva um programa que, dado a nota de um aluno mostre se ele está aprovado, de recuperação ou reprovado

```
nota = 7

```

```
if nota >= 6:
    print("Aprovado")
elif nota > 5:
    print("Recuperacao")
else:
    print("Reprovado")
```

Saída: Aprovado

Exemplo usando lógica booleana

Um aluno é considerado aprovado se sua média for maior ou igual a 6 e se sua frequência for maior ou igual a 0.75. Escreva um programa que dada a nota e a frequência de um aluno informe se ele está aprovado ou reprovado

```
nota = 7
frequencia = 0.65

if nota >= 6 and frequencia >= 0.75:
    print("Aprovado")
else:
    print("Reprovado")
```

Saída: Reprovado

7.7 Exercícios

1. Escreva um programa que leia dois valores do teclado e mostre qual o menor deles.
2. Para doar sangue, é necessário:
   - Ter entre 16 e 69 anos;
   - Pesar mais de 50kg
   - Estar descansado (ter dormido pelomenos 6 horas nas últimas 24 horas)

   Faça um programe que pergunte a idade da pessoa, o peso e quanto dormiu nas últimas 24 horas e informe se ela pode doar sangue ou não.
3. Considere uma equação do segundo grau: $ax^2 + bx + c$. A partir dos coeficientes,

determine se a equação possui duas raízes reais, uma raíz real ou se não possui.

Dica: $\Delta = b^2 - 4ac$: se delta é maior que 0, possui duas raízes reais; se delta é 0, possui uma raiz; caso delta seja menor que 0, não possui raiz real.

4. Leia dois números e efetue a adição. Caso o valor somado seja maior que 20, este deverá ser apresentado somando-se a ele mais 8; caso o valor somado seja menor ou igual a 20, este deverá ser apresentado subtraindo-se 5.
5. Leia um número e imprima a raiz quadrada do número caso ele seja positivo ou igual a zero e o quadrado do número caso ele seja negativo.
6. Leia um número inteiro entre 1 e 12 e escreva o mês correspondente. Caso o usuário digite um número fora desse intervalo, deverá aparecer uma mensagem informando que não existe mês com este número.
7. Você já não aguenta mais responder "Bom dia", "Boa tarde" e "Boa noite" nos grupos do whatsapp da família, por isso, resolveu fazer um programinha que, dada a hora correspondente do dia, manda a resposta certa. Entre as dezoito (seis da tarde) e as seis da manhã, faça seu programa mandar "Boa noite", das seis da manhã ao meio dia, "Bom dia", e do meio dia às dezoito horas, "Boa tarde". Considere que não serão passados números fora do intervalo de 0 à 23.
8. Dada a seguinte tabela:

Temperatura as 8h	Umidade do ar	Itens de sobrevivência
$10 < 15$	$0 < 40$	Blusa de frio e regata
$10 < 15$	$40 <= 100$	Blusa de frio, regata e guarda-chuva
$15 < 20$	$0 <= 40$	Blusa de frio e regata
$15 < 20$	$40 <= 100$	Blusa de frio, regata e guarda-chuva
$20 < 25$	$0 < 40$	regata
$20 < 25$	$40 <= 100$	regata e guarda-chuva

Compare se a temperatura se encaixa em um dos intervalos explicitados e depois verifique a umidade do ar para classificar. Caso a temperatura não esteja em nenhum dos intervalos, imprima a mensagem "Melhor nem sair de casa hoje!".

8. Estruturas de Repetição

8.1 Para que serve?

A estrutura de repetição permite que um trecho de código seja executado mais de uma vez, de acordo com uma condição ou com um contador.

Imagine que você queira escrever um programa que imprima os números de 1 a 10. Com os conhecimentos adquiridos até agora isso é possível e relativamente fácil, mas e se aumentarmos o número para 100 ou até mesmo 1000? Esta tarefa fica inviável, porém possível. No entanto, se pedirmos para o usuário informar um número e tivermos que imprimir de 1 até o número dado, esta tarefa fica impossível de ser realizada com os conhecimentos atuais.

As estruturas de repetição tornam essa tarefa possível e simples. Existem dois tipos principais de estruturas de repetição em Python, que serão mostrados em seguida.

8.2 For (para)

8.2.1 Quando devo usar?

O *for* é usado principalmente quando é preciso executar um conjunto de comandos uma quantidade X de vezes.

8.2.2 Funcionamento

Existe um contador (por convenção é chamado de i, j ou k) que vai de um valor inicial dado até um valor final dado, em um passo x. Em seguida um bloco é iniciado executando um comando.

A seguir é mostrado a sintaxe do for em Python:

```
for i in range(inicio, final, passo):
    comandos
```

O i é uma variável que assume o valor inicial e a cada loop é incrementada no valor do passo até que ela seja menor do que o valor final. Se i for igual ou maior que o valor final o loop é interrompido e o interpretador passa para a próxima parte do programa.

Quando o início não é especificado, o interpretador assume que o valor é 0 e quando o passo não é especificado, ele assume que é 1. Assim, se dentro do parênteses da função $range()$ só possuir um valor, será o valor final e se possuir dois valores será o valor inicial e o valor final.

Note que assim como a estrutura condicional o for também é indentado, ou seja, após os dois pontos é necessário dar um *tab* em todas as linhas do comando para que o interpretador saiba que elas pertencem a um mesmo bloco.

Exemplo 1:

Faça um programa que imprima a string "oi" dez vezes na mesma linha, separadas por um espaço.

```
for i in range(10):
    print("oi", end=" ")
```

Saída: oi oi oi oi oi oi oi oi oi oi

Neste programa, o i começa com o valor 0 e vai até ser menor que 10. A cada vez que ele faz o loop ele realiza os comandos, que nesse caso é imprimir a função print.

Exemplo 2:

Faça um programa que imprima os números de 1 a 10 na mesma linha, separados por um espaço.

```
for i in range(1,11):
    print(i, end=" ")
```

Saída: 1 2 3 4 5 6 7 8 9 10

Neste programa, o valor de i vai de 1 até ser menor que 11 e a cada loop ele imprime o seu valor.

Exemplo 3:

Faça um programa que imprima os números ímpares 1 a 10 na mesma linha, separados por um espaço.

```
for i in range(1,11, 2):
    print(i, end=" ")
```

Saída: 1 3 5 7 9

Neste programa, o valor de i vai de 1 até ser menor que 11, sendo incrementado em 2 e, a cada loop ele imprime o seu valor.

Exemplo 4:

Crie um programa que leia um texto e mostre quantas letras "a" tem no texto.

```
texto = input()
tamTexto = len(texto)

count = 0
for i in range(tamTexto):
     if texto[i] == 'a':
             count = count + 1

print(count)
```

Para esse exemplo, começamos lendo o texto como entrada. Em seguida, pegamos qual o tamanho do texto com a função $len()$ e guardamos na variável $tamTexto$. Setamos o nosso contador ($count$) para 0 e começamos o for. O i vai de 0 até o tamanho do texto e para cada loop, é comparado o caractere do texto da posição i com a letra a, se for igual ele adiciona no

contador e se não for igual ele não faz nada. Ao final, após sairmos do loop, imprimimos o valor do contador para sabermos quantas vezes aparece a letra "a" no texto.

8.3 While (enquanto)

8.3.1 Quando devo usar?

O while é normalmente utilizado quando um bloco de código precisa ser repetido enquanto uma condição permanecer verdadeira. Se a condição virar falsa, o programa pula para o próximo trecho de código.

Note que ao contrário do for, o while não precisa saber exatamente quantas vezes o bloco de código será executado.

8.3.2 Funcionamento

O *while* possui uma condição que verifica alguma variável e enquanto for verdadeira, ele executa o bloco de código. É preciso também, que essa variável sofra alguma alteração dentro do *while*, para não criarmos um loop infinito.

A seguir é mostrado a sintaxe do *while* em Python:

```
1 while condicao:
2     comandos
```

Note que assim como a estrutura condicional e o *for*, o *while* também é indentado, ou seja, após os dois pontos é necessário dar um *tab* em todas as linhas do comando para que o interpretador saiba que elas pertencem a um mesmo bloco.

Exemplo 1

Faça um programa que imprima os números de um a dez.

```
1 i = 1
2 while i <= 10:
3     print(i, end=" ")
4     i = i + 1
```

No *while* para este exemplo temos que criar uma variável de controle, que fará parte da condição. Neste exemplo, nossa variável de controle é o i, e o loop roda até que i atinja o valor de 10. Para isto, dentro do loop é preciso atualizar seu valor.

Exemplo 2

Faça um programa que peça ao usuário para digitar número enquanto a soma deles for menor do que sem. Depois, imprima o resultado da soma.

```
1 soma = 0
2 while soma < 100:
3     numero = int(input())
4     soma = soma + numero
5
6 print(soma)
```

Neste caso, a variável de controle é *soma*, começamos iniciando-a com 0. A cada iteração, a soma é atualizada com o valor do número digitado no teclado. Quando essa soma for maior ou igual a 100, a condição do *while* é quebrada, assim como o loop. Em seguida, o valor da soma é printado, como requisitado pelo exercício.

Exemplo 3

Leia números do teclado até encontrar dez números pares, então, retorne a soma de todos esses números pares.

```
1  soma = 0
2  qntPar = 0
3
4  while qntPar < 10:
5      numero = int(input())
6      if numero % 2 == 0:
7          qntPar = qntPar + 1
8          soma = soma + numero
9
10 print(soma)
```

Aqui, a variável de controle é $qntPar$ que é a quantidade de números pares que foram lidos. Inicialmente ela é setada em 0 e utilizamos uma condicional dentro do loop para saber se o valor lido é par ou não. Se for, ele atualiza a quantidade de números pares lidos até então, assim como sua soma. Quando o números de pares chega a 10, o loop é interrompido e a soma é mostrada na tela.

8.4 Exercícios

Para cada exercício deste capítulo, faça uma versão usando for e uma outra versão usando $while$. Se não for possível resolver com uma das duas estruturas, explique o porquê.

1. Calcule a tabuada do 13.
2. Leia do teclado uma lista com 5 inteiros e imprima o menor valor.
3. Leia do teclado uma lista com 5 inteiros e imprima $True$ se a lista estiver ordenada de forma crescente e $False$ caso contrário.
4. Exiba em ordem decrescente todos os números de 500 até 100.
5. Leia 10 números do teclado e imprima qual a quantidade que está entre 10 e 50.
6. Leia do teclado a idade e o sexo de 10 pessoas, calcule e imprima:
   (a) Idade média das mulheres
   (b) Idade média dos homens
   (c) Idade média do grupo
7. Calcule o somatório dos números de 1 a 100 e imprima o resultado.
8. Leia uma lista com 10 números e imprima-a em ordem crescente.

9. Funções

9.1 O que é uma função?

Muitas vezes, quando estamos programando, precisamos fazer um mesmo bloco de código mais de uma vez. Por exemplo, vamos supor que você está fazendo um programa em Python que precisa calcular a média de um aluno antes e depois da prova de recuperação. Para este caso, você precisaria escrever a fórmula da média duas vezes, e se o cálculo da média mudar, você teria que alterá-lo no seu código duas vezes também. Em larga escala, além de ser cansativo escrever o mesmo bloco de código diversas vezes, erros tornam-se mais suscetíveis e alterações mais difíceis de fazer.

Para solucionar este problema, utilizamos *funções*. Uma função é uma sequência de comandos que executa uma tarefa e que tem um nome. Ela permite deixar seu código organizado em blocos, sendo que cada bloco executa um determinado tipo de código.

Você pode não ter percebido, mas ao longo do livro vimos diversas funções o tempo todo, como por exemplo a *len*(), *print*(), *input*(), entre diversas outras. Nós a chamamos para que realiza-sem determinada tarefa Essas funções citadas já existem dentro do Python, neste capítulo vamos aprender como criar outras.

9.2 Definindo uma função

Para definir uma função, é necessário ter um nome para ela e quais são os parâmetros necessários. A sintaxe de uma função é a seguinte:

```
def nome_funcao(parametros):
    bloco de comandos
```

Exemplo 1

```
def hello(nome):
    print("ola, {}".format(nome))
```

Esta é uma função que recebe um nome como argumento e quando chamada, ela printa na tela a mensagem $ola, nome$. Perceba que ela não retorna nenhum valor.

Exemplo 2

Vamos supor que você queira fazer uma função para calcular o quadrado de um número. Ela ficaria desta forma.

```
def quadrado(numero):
    return numero*numero
```

Esta função possui o *return*, ou seja dada uma ou mais entradas (parâmetros) ela retorna um valor.

9.3 Chamando uma função

Repare que se rodarmos os programas dos exemplos anteriores, nada acontecerá. Isso acontece porque a função não foi chamada, ela está apenas definida.

Para chamar uma função, basta escrever seu nome e entre os parênteses colocar os valores desejados. As variáveis dos parâmetros irão receber estes valores na ordem em que foram digitados. Se a função possuir retorno, é possível também armazenar este valor em uma variável.

Exemplo 1

Vamos considerar o exemplo da seção anterior:

Função:

```
def hello(nome):
    print("ola, {}".format(nome))
```

Programa completo:

```
hello('Fernanda')

def hello(nome):
    print("ola, {}".format(nome))
```

Saída: ola, Fernanda

Neste caso, a variável *nome* recebe a string *Fernanda*.

Exemplo 2

Função:

```
def quadrado(numero):
    return numero*numero
```

Programa completo:

```
numero = int(input())
quad = quadrado(numero)

print("O quadrado de {}     {}" .format(numero,quad))

def quadrado(numero):
    return numero*numero
```

Neste caso, armazenamos o valor de retorno em uma variável e depois mostramos na tela uma frase que fala qual o número que foi inserido e qual o seu quadrado.

9.4 Exercícios

Resolva todos os exercícios abaixo usando funções.

1. Dado dois números a e b, retorne a sua soma.
2. Escreva um programa que recebe a base e o expoente e calcule a exponencial do número.
3. Escreva um programa que recebe um número e informe se ele é par ou impar.
4. Dada a fórmula de Bhaskara:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{9.1}$$

   Escreva um programa que dados valores de a, b e c, calcule os possíveis valores de x.
5. Alanderson quer saber se um endereço IP é válido. Faça um programa para ajudar Alanderson a testar se um endereço é válido.

   Para isso, a entrada deve ser um endereço IP (digitado pelo usuário) e o programa deve escrever na tela se é válido ou não. Um endereço IP é composto por 4 números inteiros entre 0 e 255, separados por um ponto.

   Por exemplo, o endereço 123.123.123.123 é válido, mas 666.123.k.3 não é.
6. Dada a função: $y = 5x + 2$, determine os valores de y para x entre -10 a +10, onde x é inteiro.

10. Exercícios e Desafios