

Apostila

Python

Progressivo



www.pythonprogressivo.net

Sobre o e-book Python Progressivo

Antes de mais nada, duas palavrinhas: parabéns e obrigado.

Primeiro, parabéns por querer estudar, aprender, por ir atrás de informação. Isso é cada vez mais raro hoje em dia.

Segundo, obrigado por ter adquirido esse material em www.pythonprogressivo.net. Mantemos neste endereço um *website* totalmente voltado para o ensino da linguagem de programação Python.

O objetivo dele é ser bem completo, ensinando praticamente tudo sobre Python, desde o básico, supondo o leitor um total leigo em que se refere a absolutamente tudo sobre computação e programação.

Ele é gratuito, não precisa pagar absolutamente nada. Aliás, precisa nem se cadastrar, muito menos deixar e-mail ou nada disso. É simplesmente acessar e estudar.

E você, ao adquirir esse e-book, está incentivando que continuemos esse trabalho.

Confiram nossos outros trabalhos:

www.programacaoprogessiva.net

www.javaprogressivo.net

www.cprogressivo.net

www.htmlprogressivo.net

www.javascriptprogressivo.net

www.excelprogressivo.net

www.cmmprogressivo.net

Certamente, seu incentivo \$\$ vai nos motivar a fazer cada vez mais artigos, tutoriais e novos sites!

Este e-book é composto por todo o material do site. Assim, você pode ler no computador, no tablet, no celular, em casa, no trabalho, com ou sem internet, se tornando algo bem mais cômodo.

Além disso, este e-book contém mais coisas, mais textos e principalmente mais exercícios resolvidos, de modo a te oferecer algo mais, de qualidade, por ter pago pelo material.

Aliás, isso não é pagamento, é investimento. Tenho certeza que, no futuro, você vai ganhar 10x mais, por hora trabalhada, graças ao conhecimento que adquiriu aqui.

Proprietário da Apostila

Esse e-book pertence a:

Cláudia Pereira Gonçalves
claudiapereira8123@gmail.com
Rua Genebra 430
Calafate
Belo Horizonte - MG
30411560
Brasil

Pedimos, encarecidamente, que não distribua ou comercialize seu material. Além de conter suas informações, prejudica muito nosso projeto.

Se desejar indicar o Python Progressivo para um amigo, nosso site possui todo o material, de forma gratuita, sem precisar de cadastro e o acesso dessas pessoas também ajuda a manter o site no ar e criamos cada vez mais projetos:

www.pythonprogressivo.net

Sumário

[Como começar a programar em Python ?](#)

[O que é o Python ? Para que serve? Onde se usa? É a melhor linguagem?](#)

[Baixar, Instalar e Rodar o Python](#)

****** [Como ser o melhor Programador Python](#)

****** [Mercado de Trabalho](#)

Básico

1. [Hello, World! Meu primeiro programa em Python](#)
2. [Função **print** – Imprimindo coisas na tela](#)
3. [Tipos de dados – Números, Strings e Booleanos](#)
4. [Função **input** – Recebendo dados do usuário](#)
5. [Funções int\(\) e float\(\) - Transformar string em números](#)
6. [Operações Matemáticas no Python](#)
7. [Exercícios Resolvidos de Porcentagem e Juros Compostos](#)
8. [Criando uma calculadora com Python](#)
9. [Precedência de operadores](#)
10. [Formatando números com a função **print**](#)
11. [Exercícios básicos](#)

Testes Condicionais

1. [Operadores de comparação: ==, !=, >, >=, < e <=](#)
2. [IF - Teste Condicional em Python](#)
3. [IF e ELSE - Instruções de teste](#)
4. [IF e ELSE aninhados](#)
5. [Exercícios de IF e ELSE](#)
6. [Instrução ELIF em Python](#)
7. [Exercício de ELIF](#)
8. [Operadores Lógicos: AND, OR e NOT](#)
9. [Exercícios finais de Testes Condicionais](#)
[Soluções](#)

Laços e Loopings: WHILE e FOR

1. [Estruturas de repetição: Entendo a lógica dos laços e loopings](#)
2. [WHILE - O que é, como funciona e como usar](#)
3. [FOR - Um loop controlado](#)
4. [A função range\(\) pro laço FOR](#)
5. [Progressão Aritmética \(PA\) com laços em Python](#)
6. [Operadores de Atribuição: += -= *= /= e %=](#)
7. [Fatorial com WHILE e FOR](#)
8. [Laços Aninhados \(Loop dentro de loop\)](#)
9. [Instrução ELSE, BREAK e CONTINUE em Laços](#)

10. Exercícios de laço FOR e WHILE

Soluções

Funções

1. Função: O que é? Para que serve? Onde são utilizadas?
2. Como declarar, chamar e usar funções: def
3. Função chamando função
4. Variáveis locais
5. Parâmetro e Argumento em Funções Python
6. Passagem por valor
7. Argumentos Posicional e Nomeado
8. Variável global
9. Constante global
10. O comando return - Como retornar valores
11. Recursividade

Exercícios de funções

Módulos

1. O que é? Para que serve? Onde se usa?
2. Como Criar, Importar e Usar um módulo: import
3. Como gerar números aleatórios em Python
4. Módulo math - Funções Matemáticas Prontas

Jogo em Python: Adivinhe o número

Listas

1. Listas em Python - O que são ? Para que servem ? Por que usar ?
2. Como criar uma lista e acessar seus itens
3. Como Usar Listas: Adicionar, Mudar, Concatenar e Outras Operações
4. Listas com Laço For
5. Matriz: Lista dentro de lista
6. Jogo da velha em Python

Tuplas

1. O que é? Para que serve? Como usar ? Quando usar?

Dicionário

1. O que é ? Para que serve? Como e quando se usa ?
2. Como exibir items, valores e chaves de um dicionário
3. Como usar os métodos get() e setdefault()
4. Adicionar, Alterar e Remover items de um dicionário
5. Exercício resolvido: Como usar Dicionários em Python
6. Como alterar o nome de uma chave (key) de um Dicionário
7. Como copiar uma Lista ou Dicionário

Arquivos

1. [Como Abrir e Ler um arquivo: open\(\) e read\(\)](#)
2. [Como ler um arquivo linha por linha: readlines\(\)](#)
3. [Como escrever em Arquivos: write\(\) - Modos de abertura](#)
4. [Como retirar a quebra de linha \(caractere '\n'\) de Arquivos](#)
5. [Processando arquivos grandes com Laço FOR](#)
6. [Módulo os: Caminhos, Endereços, Arquivos, Pastas e Diretórios](#)
7. [Como ler e escrever ao mesmo tempo num mesmo arquivo](#)

[Lista de exercícios de Arquivos](#)

Strings

1. [O que é? Como funciona ? Para que serve ? Onde vamos usar ?](#)
2. [Como usar, Acessar caracteres e Descobrir tamanho de string](#)
3. [Strings maiúsculas e minúsculas](#)
4. [String só de letra, número e/ou caractere especial](#)
5. [Juntar e Separar strings: join\(\) e split\(\)](#)
6. [Como Localizar e Substituir algo em uma string](#)
7. Expressões regulares:
 - Parte 1: [Introdução, Módulo re, recompile, search, objetos Regex e Match](#)
 - Parte 2: [Grupos e parêntesis](#)
 - Parte 3: [Caractere Pipe | \(o OU\)](#)
 - Parte 4: [Ponto de interrogação \(?\), Asterisco \(*\) e Soma \(+\)](#)
 - Parte 5: [Classes de Caracteres](#)
 - Parte 6: [Início \(^\) e Final de String \(\\$\)](#)
 - Parte 7: [Caractere curinga - Ponto .](#)
 - Parte 8: [Substituindo strings com o método sub\(\)](#)
 - Parte 9: [Como fazer regexes longas e complexas](#)

Orientação a Objetos

1. [O que é Classe e Objeto](#)
2. [Como criar uma Classe e um Objeto](#)
3. [Métodos, __init__ e Atributos](#)
4. [Parâmetros e Argumentos de métodos em classes](#)
5. [Método Construtor __init__ : Parâmetros e Argumentos](#)
[Jogos: Cara ou Coroa e Lançamento de dados](#)
6. [Atributos privados](#)
7. [Atributos de classe](#)
8. [Composição: Objeto dentro de Objeto](#)
9. [Herança: O que é ? Para que serve?](#)
10. [Como usar herança - Superclasse e Subclasse](#)
11. [Polimorfismo em Python](#)

GUI - Programação Gráfica

1. [O módulo Tkinter](#)
2. [Label - Exibindo rótulos \(textos\) numa janela](#)
3. [Frame - Widget que armazena widgets](#)
4. [Caixas de diálogo - info dialog boxes](#)
5. [Botões - o widget Button](#)
6. [Entry widget - Recebendo entradas do usuário](#)
7. [StringVar - Saída dinâmica de dados com label](#)
8. [Botão de rádio - Radiobutton e IntVar](#)
9. [Botão de check - Checkbutton](#)
10. [Tratamento de eventos: classe Event e método bind\(\)](#)
11. [Dando enter: o evento <Return>](#)
12. [Eventos envolvendo botão: <Enter> e <Leave>](#)
13. [Tratamento de eventos envolvendo Mouse](#)
14. [Tratamento de eventos envolvendo Teclas do teclado](#)
15. [Gerenciadores de Layout: pack, grid e place](#)

Como ser o melhor programador Python Possível

A partir do próximo tutorial, você vai mergulhar profundamente no apaixonante mundo da programação. É sem volta, mesmo que use profissionalmente ou não. Na pior das hipóteses, vai apenas te deixar mais inteligente, com melhor criatividade e raciocínio mais rápido.

Como estudar programação

Infelizmente, não é só ler ou ver um vídeo, como muitos assuntos por aí. Programação faz parte da Computação, que faz parte da Matemática. Ou seja, programar é um exercício de lógica puro.

Mas não se assuste, sei que programação tem aquela áurea de 'ohh, nossa que fodástico, só gênio pra fazer isso', mas isso é totalmente falso, é possível qualquer pessoa aprender TRANQUILAMENTE, a programar.

Mas vou ser honesto com você: precisa de muito esforço, muita dedicação, e ser uma pessoa insistente.

Dá vontade de desistir, eu mesmo já larguei por diversas semanas, várias vezes, em vários assuntos e várias linguagens.

Leia nossa apostila, pesquise por tutoriais, veja vídeo-aulas...não entende? Estude de novo, leia de novo, pense de novo...as vezes, nem assim vai. Depois você volta no tópico, é assim mesmo, devagar, sem pressa.

É de extrema importância você escrever o código e ver ele rodar na sua frente, diante de seus olhos. Quer apenas ou apenas ver vídeo-aulas? Vai perder seu tempo. Não existe absolutamente **nenhum** programador que só leu ou assistiu aulas.

Você só aprende, única e exclusivamente, se você digitar os códigos, entender eles. O código tem que rodar na sua cabeça antes, fazer sentido para você, só então depois que você vai começar a digitar e ver ele funcionando.

E mesmo assim, vai errar MUITO, mas MUUUUUUU...UUUUUU MESMO. Eu errei. Você vai errar, todo mundo erra muito. A gente fica louco, caçando os erros, quebrando a cabeça, pesquisando e chorando em posição fetal por causa dos erros. E as vezes é porque esquecemos um ponto-e-vírgula, apenas.

Outra coisa extremamente comum que vai acontecer com você: escrever códigos longos, confusos e cheio de 'remendos'. É assim mesmo que

funciona. Aí você vai ver o código aqui da apostila, está feito em 20 linhas o que você fez em 200 linhas.

PARABÉNS pelas 200 linhas! É com essa força de vontade e perseverança que você vai se tornar um programador EXCELENTE. Da próxima vez que for refazer esse código de 200 linhas, já vai facilmente fazer em 150...depois 100...e logo logo tá fazendo um código melhor que a gente.

É ASSIM MESMO QUE FUNCIONA, ok ?
Não desanime, vai ser complicado mesmo.

O que estudar ?

Essa é outra dúvida que me assombrava...Java, C, C++, Python, Redes, Sistemas Operacionais, Administração de Redes, Banco de dados...ahhh, eu ficava louco e queria aprender tudo, e acabava aprendendo nada direito.

Pessoal, é sem pressa. NUNCA ESTUDE COM PRESSA.

A resposta sobre por onde começar é: essa apostila. Ela já assume que você é totalmente leigo de tudo, só sabe ligar o computador, abrir essa apostila e pronto. Ensinamos até onde você vai clicar e digitar, para começar a programar.

Mas esse é só o início da jornada, ok?
Você deve estudar outras linguagens, outros paradigmas.

Por exemplo, com Python, você tem uma linguagem de script, pra escrever pouco e fazer muita coisa simples, é ótimo pra otimizar suas tarefas no trabalho e até suas coisas pessoais.

Hoje o mundo gira em torno dos apps. Com Java, você vai ser apto a criar aplicativos Android e softwares que rodam em várias plataformas (Windows, Linux...).

C e C++ são usados para criar programas extremamente eficientes, rápidos e confiáveis, você tem um poder absurdo sobre a máquina, e muita responsabilidade em mãos.

Com o trio HTML (não é linguagem de programação) + JavaScript + PHP você simplesmente pode quebrar a internet. A grande rede é montada em cima dessas linguagens, e sabendo elas, você cria qualquer sistema web.

Quer realmente entender o que é um bit? O que é 0 e 1? Controlar cada byte de informação da sua máquina, mover eles quase que manualmente de um registro pra outro da sua CPU? Estude Assembly.

E por ai vai, pessoal. Cada linguagem e tecnologia nova que você aprender, você fica mais inteligente, mais criativo e com uma linha de raciocínio beeem melhor, se tornando um programador cada vez mais completo. Não se limite.

Qual a melhor linguagem de programação?

“Ah, com Java eu crio apps pro Android facilmente”

Então Java é melhor? Cria então um sistema operacional em Java...totalmente inviável, Java é um pouco mais ‘pesado’.

“Ah, Linux foi feito em C e Windows foi feito em C++, logo são as melhores linguagens.” Ok bonitão, então vai lá e tenta criar um aplicativo pra Android com essas linguagens, vai demorar 10 anos pra fazer o que o Java faria em 1 mês.

“Por isso prefiro Python, super leve, escreve poucas linhas e é extremamente versátil”. É fácil fazer muita coisa em Python, né? Então faz um ‘hello, world’...mas na tela da BIOS, ou seja, naquela telinha preta que aparece quando você inicia seu computador. Ali só vai com C ou Assembly.

“Então, obviamente Assembly é a mais foda de todas, pois é possível fazer tudo com ela”. Sim, teoricamente é possível fazer tudo com ela, então tenta criar uma rede social com ela. Mas vai precisar contratar mil programadores pra terminar dentro de uma década.

Porém, com PHP e Banco de Dados, por exemplo, numa tarde você pode criar uma baita rede social e deixar ela bem interessante usando bastante JavaScript.

O que quero dizer com isso?

Que não existe melhor linguagem de programação. Existem linguagens mais adequadas para cada tipo de problema. Repita as duas frases anteriores. Se possível, imprima e cole na parede atrás do seu computador.

Pessoal, isso inclusive é muito perguntado em entrevistas de emprego. Se vier com esse papinho de “Java é a melhor”, “Python rules”, “PHP é a mais usada”, provavelmente não vão te contratar.

A coisa mais normal do mundo é uma empresa mudar de seus sistemas de linguagens e de tecnologias. É aí que entramos no próximo tópico...

A coisa mais importante para se aprender é...

...aprender.

A melhor característica de um programador é saber aprender. Aprenda a aprender.

Não importa seu nível, quantas línguas saiba, quantos anos de experiência tem...você sempre vai precisar fazer uma pesquisa no Google para lembrar algum detalhe.

Essa característica, no programador, de saber buscar informações e aprender rápido, é sem dúvidas a mais importante.

Você precisa ser flexível, estar preparado para estudar outra linguagem, usar outra tecnologia...não se limite, não seja fanático por linguagem, tecnologia ou sistema, seja aberto para as novidades.

Continue sempre pesquisando, seja curioso, teste as coisas, duvide dos códigos, tente fazer melhor, pense numa solução melhor enquanto estiver tomando banho ou antes de dormir.

Como começar a trabalhar com programação?

Assim que uma pessoa termina de estudar algo, mesmo sendo uma graduação, ela ainda não está preparada para trabalhar plenamente, a ponto de dar lucro e ser interessante para uma empresa.

Um engenheiro civil não sai apto a construir uma ponte enorme ou um arranha-céu. Porém, ele é capaz de aprender isso com alguns meses ou anos de experiência, com outras pessoas que sabem.

Se quer trabalhar na área, indico algum site de freelancer e redes sociais, de freelancer. E ser muito cara-de-pau, não pode ter vergonha.

Se ofereça para ajudar um programador profissional, de graça mesmo. É, eu sei que todo estagiário hoje em dia já quer salário completo, alimentação, transporte e tapinha nas costas de cara. Mas a realidade do mundo é outra, eu mesmo passei mais de ano trabalhando sem ganhar nada.

Aliás, ganhei sim, a coisa mais valiosa: experiência. Faça projetos, crie jogos, crie programas úteis, tente se aproximar de outros programadores experientes, que já estão no mercado de trabalho.

Se entendeu e aceitou tudo que eu disse acima (ah se eu tivesse alguém pra ter me dito isso quando eu comecei, seria um programador muuuuito melhor!!!), você já está apto a começar aprender a programar.

Estude com calma, leia, releia, pesquise, questione e sempre tente mais uma vez. Não se limite a nada!

Simbora, programar?

Mercado de Trabalho

De todas as inúmeras perguntas e dúvidas que recebemos aqui diariamente, sem dúvidas a mais recorrente é:

Como me tornar um programador Python profissional, trabalhar com isso?

E não é de se surpreender, Python é uma linguagem fantástica, simplesmente linda, apaixonante, simples de aprender, usar e **absurdamente** poderosa.

Se estudar e programar Python já é bom, imagina:



- Ganhar dinheiro com Python
- Ter seu apartamento ou casa, graças ao Python
- Ter um carro bacana, seguro e bonito, com dinheiro vindo da programação Python
- Fazer viagens, conhecer o mundo e novas culturas, graças ao seu trabalho com Python

E **sim**, isso é possível!

Faça seu café, sente-se confortavelmente e tire 15min do seu dia para ler algo que pode mudar sua vida.

- **Programador Python: Tem mercado? Quem precisa ?**

Você tem amigos que cursam ou cursaram direito?

Eu tenho (muitos, diga-se de passagem).

Pergunte a eles como está o mercado, o salário...e prepare-se para histórias tristes. É muito, muito difícil conseguir um emprego para quem é formado em cursos como Direito, Administração etc. Nada contra, pelo amor de deus.

Mas o motivo é bem simples: todo bairro hoje em dia tem uma faculdade que oferece esses cursos. E o número de formandos é imenso. Todo santo semestre dezenas de milhares de alunos se formam nessas áreas...e infelizmente **não tem vaga pra todo mundo**.

Até pra medicina, curso notavelmente conhecido por seu bom salário e a existência de vagas, a coisa está complicando. O salário médio de um médico vem diminuindo, é algo extremamente difícil conseguir uma residência pra se especializar...e muitos tem que ir pra interior, e todos sem exceção, fazem jornadas árduas e longas de trabalho.

E não é pra menos, cada vez mais faculdades, formandos, situação ruim do país...



E pra Python, como está o mercado?

Quantos advogados, engenheiros, estudantes de medicina, economia, administração etc, você conhece? Muitos, aposto.

E programadores? Se conhecer um, já é muito.

Agora outro exercício:

- Olhe pra suas mãos: celular, tablet, notebook...
- Olhe pra frente: notebook, TV smart, painel digital de um carro...
- Olhe ao redor: em supermercados, ruas, farmácias, pro céu...

Onde **não tem** programação? Duvido achar um local que não tenha algo com algum software. Pode ir em uma favela ou sertão nordestino que vai ver gente com algum dispositivo digital.

O aumento do uso da tecnologia é MEDONHO. A carência, necessidade de gente nessa área é **MONSTRA**.

E o tanto de gente nessa área você respondeu: é pouco!

Raro encontrar um programador, alguém que faz isso, vive disso...e não é pra menos, não é algo tão simples.



[Quem usa Python?](#)

O segredo da programação no Mercado de Trabalho

Cada vez mais a tecnologia vai estar presente no mundo, inclusive substituindo humanos.

E quem vai sair na vantagem? Quem é da área.

Vão precisa de fazer pra fazer, criar, organizar, ensinar...coisas relacionadas a tecnologia e programação.

Sempre. O mundo vai ser dos programadores, a verdade é essa. Pura, simples e clara.

Desde grandes companhias aéreas, fábricas de carros até o botequim do seu Zé vai precisar de tecnologia, softwares....amigos, tudo, em todo canto, é sério.

Ao contrário de advogados e administradores, que não a demanda é menor e o tanto de gente estudando é cada vez maior, no mundo da programação é o oposto: cada vez mais precisamos de programadores e tá cada vez mais raro achar um.

Não estou tentando de convencer ou 'vender meu peixe', até porque não vamos ganhar nada com isso. Apenas olhe ao seu redor e tire suas conclusões.



```
set1 = self.filter(from_user=user).select_related(depth=
set2 = self.filter(to_user=user).select_related(depth=
return set1 | set2

def are_connected(self, user1, user2):
    if self.filter(from_user=user1, to_user=user2).count() > 0:
        return True
    if self.filter(from_user=user2, to_user=user1).count() > 0:
        return True
    return False

def remove(self, user1, user2):
    """
    Deletes proper object regardless of the order of users in
    """
    connection = self.filter(from_user=user1, to_user=user2)
    if not connection:
        connection = self.filter(from_user=user2, to_user=user1)
    connection.delete()
- models.py Top L1 (Python AC vns)
```

- **Como Ser Programador Python**

Tem certeza absoluta da importância e que nunca vai faltar trabalhos para um programador? **Excelente.**

Vamos mais além: dos programadores, a gigantesca maioria ou sabe PHP ou Java. Nada contra. Mas ache um programador Python, uma linguagem em crescimento exponencial nos Estados Unidos, Japão e Europa. Não acha.

E quem sabe Python, sabe criar sites, fazer aplicativos, serve para trabalhos acadêmicos, etc etc etc. Aprende uma coisa, depois vai ter é dificuldade de qual área seguir, de tantas possibilidades.

Então vamos te ensinar agora **como ser um programador Python**. São apenas dois passos.

- **Passo 1 para ser Programador:**
Estude insanamente

Não tem pra onde correr.

Se perguntar aos melhores engenheiros, médicos, físicos, programadores, empresários e tudo mais que imaginar, o que fizeram diferente, a resposta vai ser: me esforcei mais que a maioria.

Sinto te informar, mas fazendo o que todo mundo faz, o tanto que todo mundo faz, você vai ser só mais um.

Quer ser realmente bom, foda, viver bem, ajudar sua família, viajar e tudo mais? Faça mais que a maioria.

- Enquanto a maioria estiver vendo série, estude e programe.
- Enquanto a maioria tá vendo tv e BBB, estude e programe.
- Enquanto a maioria tá dormindo até meio-dia, estude e programe.
- Enquanto a maioria tá perdendo tempo, fazendo nada e vendo o tempo passar, estude e programe.

Não tem segredo nem mistério, é estudar muito, o máximo possível. De noite, no ônibus, dar uma lidinha antes de dormir etc etc.

Pra fazer isso, a gente tá aqui pra te ajudar.

Se estudar pelo [Curso Python Progressivo](#), vai ter um curso completo, bem explicado, com muitos exemplos, exercícios, projetos reais, indicação de vídeos, livros...CONSUMA TUDO!

Estude e tente fazer os programas até chorar em posição fetal (isso acontece muito, até durmo assim vez e outra, de tão acostumado).

Essa é a parte mais importante, é 99% do segredo para se tornar um bom programador Python profissional, é a mais difícil, trabalhosa e custosa.

A segunda parte é, de longe, a mais fácil, simples e barata.

• **Passo 2 para ser Programador:** **Obtenha um certificado**

Você pode ser o médico mais foda do universo, mas só vai trabalhar se for formado e tiver seu diploma.

Mesmo Einstein, pra dar aula de Física, teve que se formar e obter certificado (ele odiava aula, faculdade e teve muita dificuldade pra se formar, de tão chato que achava).

Não importa o que vai fazer, é preciso que, digamos, você 'prove' que estudou. No caso do nosso mundo da programação e computação, isso vem por meio de certificado e diploma.

Se não faz faculdade, não tem tempo, dinheiro ou mesmo curso na sua cidade, **relaxe totalmente**. Um bom programador é reconhecido por seu talento, seu código, seu software, e não se é formado na Uniesquina, ITA ou curso online.

Então, dou aqui a solução pra vocês:



[Clique para obter sua certificação](#)

- **Curso de Python com Certificado:**

Por que fazer ?

Bom, vamos resumir o motivo pela qual você deve fazer o curso acima, o mais rápido possível:

- **Preço:** é absurdamente barato, chega a ser ridículo o preço. Pagamento único. Sim, tem que meter a mão no bolso, eu meti, todo mundo que vive de Python já pagou e não existe outro jeito de ter sua certificação
- **Tempo de aula:** são 72 horas / aula, de MUITO conteúdo, com uma avaliação completíssima ao final
- **Acesso:** só precisa ter internet. Pode acessar do computador, tablet, celular, no ônibus, no intervalo da escola/faculdade/trabalho. Tá sempre online.

- **Disponibilidade:** 24 horas. Qualquer dia, qualquer horário, dia de semana, final de semana, feriado...estude quando quiser e quando puder (é aqui que a galera de faculdade sente inveja)
- **Ritmo:** demore o tempo que quiser, em qualquer aula, explicação, exercícios etc. Faça tudo no seu tempo. Só tem 20 min durante o almoço? Pode ter seu certificado. Só 1h antes de dormir? Opa, aqui mesmo que vai obter seu certificado. Demora pra aprender? Vá devagar, repita...Esqueceu? Volte e estude de novo
- **Material:** o curso inteiro é bem organizado e dividido por etapas. Tem muita questão resolvida, muito exercício, muito código comentado, criação de softwares e jogos
- **Certificado:** Serve para completar Horas em Atividades Extracurriculares, em Universidades. Pode contar como Atividades em Concursos Públicos. Pode constar em seu Currículo.



[Clique aqui para obter seu certificado e entrar no mercado de trabalho](#)

Como começar a programar

Sempre que alguém fala em programador, cientista da computação, engenheiro de software e coisas do tipo, no imaginário popular vem logo a imagem de alguém *nerd*, gênio ou vulgo CDF (como chamamos aqui no nordeste).

Não, não precisa ser um gênio pra isso.

Não precisa tirar só 10 na escola pra aprender a programar

Não precisa ir pra faculdade pra aprender computação

Aliás, sabia que muito dos gênios desse ramo, abandonaram a faculdade?

Muitos donos de empresa de tecnologia, começaram programando, estudando sozinho e hoje são muito bem sucedidos, e alguns só tem ensino médio?

Pois é, se essa galera, antigamente, sem internet, sem Youtube, sem e-books e PDF, aprenderam, não tenha dúvidas: você pode e vai aprender também.

Não precisa ser inteligente nem ter conhecimento prévio algum.

Mas depois de aprender a programar, se prepare: sua mente vai mudar completamente.

O raciocínio muda.

A criatividade muda.

Sua lógica muda.

Sua mente muda totalmente, é algo incrível.

Mas vamos te dar algumas orientações, talvez não goste de algumas, mas te falo com sinceridade e *mando a real na lata*:

1. Você precisa se esforçar. Programar é simples, mas não simplório. Vai precisar ler, reler, pensar, pensar de novo, tentar e tentar. Isso, as vezes, é desgastante.

2. Precisa arranjar tempo. Muitos que querem programa estão na escola, faculdade, outros até trabalham e tem família. Você vai precisar de tempo. Boa parte do que aprendi, foi estudando em ônibus (não tinha *smartphone*, imprimia mesmo os livros e ia lendo no ônibus).

3. Precisa de sacrifícios. As vezes da vontade de ver uma série na Netflix, mas vá estudar. As vezes dá vontade de ficar dando *refresh* no Instagram e não fazer nada, mas estude. Você vê um vídeo no Youtube e ele te indica 20 outros legais, eu sei como é, mas estude. Se você continuar agindo como todos, se esforçando como todos, gastando seu tempo como todos...vai ser como todos. Quer algo diferente? Precisa agir diferente, amigão.

4. Tente de novo. As vezes, você vai ver exercícios aqui que você vai ler, tentar e vai falhar. No começo, vai ser sempre. Outras vezes, vai tentar tentar...e nada. Dá vontade de ver a resposta, a solução, mas tente mais. Dê uma volta, saia, vá passear, comer algo, namorar e depois tente novamente com a cabeça fresca. Programação é isso, todo dia você vai precisar solucionar algo, tem que quebrar a cabeça, até chorar em posição fetal no banheiro. FAZ PARTE. Só assim se vira um bom programador, ok? Tentando de novo.

5. Você vai se frustrar. Você vai passar 1 mês em um projeto, vai suar, quebrar a cabeça e resolver tudo em 5 mil linhas. Depois descobre que um filho de uma mãe resolveu de maneira melhor, mais rápida, completa e eficiente em 500 linhas. Você ficar pistola, com vontade de desistir e com vontade de decorar uns livros de leis e artigos pra passar num concurso e nunca mais ter que estudar e quebrar a cabeça. Mas se parar pra estudar, ver a solução e como funciona a mente dos outros, você vai ficar cada vez mais e mais fodástico.

6. Você vai sempre precisar estudar. Não importa o quanto estude, vai sempre precisar estudar. Sempre tem uma coisa nova pra aprender, algo que ainda não sabe, sempre vai precisar ler uma documentação de alguma API ou alguma dúvida em algum fórum. Faz parte, tem ter que gana e vontade de aprender. Não se admire se estiver trabalhando programando em um empresa e quando chegar em casa tu do que vai querer é...programar naquele seu projeto pessoal.

7. Estude inglês. Não sabe? Comece. Traduza umas músicas que gosta, veja seriados com legendas em inglês e voz em português, depois reveja com áudio original e a legenda em pt-br. Se puder, faça um curso. Inglês é a língua universal, programadores brasileiros, chineses, indianos, africanos, alemães etc etc, falam em inglês, até os americanos falam em inglês. É MUITO IMPORTANTE ESTUDAR INGLÊS. E calma, não precisa se apressar, dá pra aprender aos poucos, recomendo o [Curso de Inglês da Brava Cursos](#). [Clique aqui](#) para entender melhor a importância do inglês no estudo da programação.

No mais, é sem mistério. Senta essa bunda aí na cadeira, e estuda, estuda, tenta, programa, estuda, pesquisa...e se precisar, manda sua dúvida:

programacacao.progressiva@gmail.com

Por favor, qualquer erro de código, lógica, português ou uma solução que achar melhor, nos avise. Impossível não fazer um material desses sem erros. Nos ajude a melhorar cada vez mais o material.

O que é o Python

Python é uma linguagem de programação.

Sendo um pouco mais rigoroso e específico: é uma linguagem interpretada, de alto nível e de múltiplos propósitos.

Mas, relaxa...não precisa esquentar com essas definições agora, no decorrer do curso você vai entender bem direitinho o que significa cada uma dessas coisas.

Não vamos te encher de palavras novas e conceitos bizarros aqui, mas te garanto que vai entender tudo perfeitamente com o tempo, conforme for estudando nosso curso.

O Python foi criado em 1989 e o nome é em homenagem a um grupo de humor britânico, uma espécie de Os Trapalhões deles lá, o *Monty Python*.

Ela foi criada com um propósito bem simples: ser fácil.

Se uma ideia ou lógica funciona em sua cabeça, é possível passar ela pra Python bem facilmente e fazer seu projeto virar um programa de verdade.



Para que serve o Python

Vamos pegar uma linguagem normal, como o português, para te explicar melhor o que é, de fato, uma linguagem de programação.

Você aprendeu a língua portuguesa, sabe escrever, ler, falar e entende quando falam.

O que é possível fazer com isso?

Ué, pode escrever um livro, jornal, anúncios, pode criar um roteiro pra uma novela, um site, pode ser o próximo Drummond ou Machado de Assis... ou nada. Depende de você.

O mesmo ocorrem com o Python.

Assim como a língua portuguesa, o Python é uma linguagem, que depois que você aprende, tem um mundo de opções.

Python é muito usado para criar scripts, ou seja, programas pequenos, curtos e que quebram o galho no dia-a-dia, seja em casa, no trabalho, em um servidor etc.

Por exemplo, quando inserir um pendrive, o script vai e copia todos os dados do pendrive pra uma pasta que você pré-definiu. Pronto, ele vai fazer isso por você

automaticamente, sem você precisar fazer nada.

Quer receber um alerta quando a ação da Petrobrás atingir um determinado valor? Cria um script em Python, pequeno e rapidinho, vai fazer isso por você.

Ou um script que vai ficar tentando descobrir a senha do Wi-Fi do vizinho? Tem alguma coisa tediosa e repetitiva no seu trabalho? Como preencher ou procurar algo? Aprenda Python que você vai programar um script pra fazer isso.

Não gosta do programa que reproduz mp3 e vídeos no seu computador? Que tal criar um seu, do seu jeito?

Teve uma ideia pra um novo jogo, tanto pra computador como pra celular? Pode fazer isso em Python.

Quer trabalhar com engenharia, Física, Geologia, fazer gráficos 3D, reconhecimento facial, robótica, inteligência artificial? Usa Python.

Quer criar um site, com servidor, serviços, uma rede social ou um Youtube da vida? Sim, é possível fazer isso usando Python.

Quer que o Palmeiras tenha um mundial? Ora, é só usar Py....não, pera, isso nem o Python :(

Assim, a resposta simples da pergunta "Para que serve Python?" é: pro que você quiser.

Vai depender, basicamente, do quanto você estiver disposto a estudar e se esforçar.

Se não estudar, ficar enrolando, não fizer exercícios nem se dedicar, certamente vai ser um péssimo programador, assim como existem pessoas que, embora tenham estudado Português, falam errado, escrevem errado e por aí vai.

Se seguir direitinho nosso curso, estudar tudo com calma, sem pressa, tentar fazer os exercícios e scripts que vamos propor, te garanto que terá uma excelente base para trabalhar com o que quiser, usando Python.

```
for num in range(2,101):
    prime = True
    for i in range(2,num):
        if (num%i==0):
            prime = False
    if prime:
        print num
```

Esse pequeno trecho código imprime a lista de números primos entre 2 e 101, ou qualquer outro intervalo que você queira.

Python - Onde é usado ?

Alguns exemplos de sites, empresas e projetos que tem código Python rodando em seu sistema:

- Google
- Dropbox
- Youtube
- Instagram
- Quora
- Spotify
- Nasa
- Yahoo Maps
- BitTorrent
- Reddit
- Mozilla Firefox

Mas mais importante do que 'onde é usado Python,' é onde ele vai ser usado: onde você quiser. Pra fazer todas suas ideias, projetos e necessidades.

Até iria brincar e dizer 'Python é usado pra fazer café', mas não vou dizer, pois vai que alguém descobre um jeito de usar ele pra isso mesmo.

Não me surpreenderia.

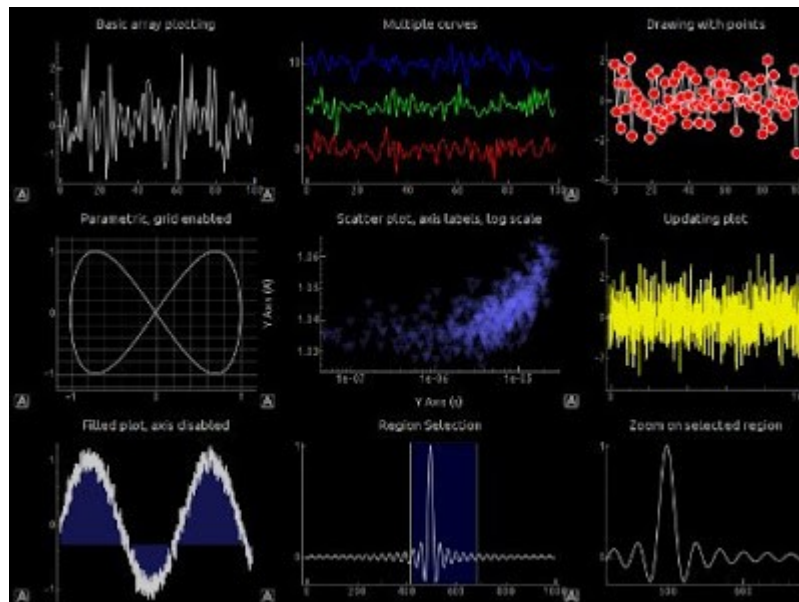
Python - É a melhor linguagem para começar a programar?

C, C++, Java, C#, PHP, Perl, Ruby...há muitas, mas muitas linguagens de programação mesmo. Então

"Por que escolher Python?"

Porque ele é simples. É, de longe, a maneira mais fácil, rápida e intuitiva de aprender a programar, sem te limitar, te permitindo criar desde scripts simples pra organizar suas coisas até websites, jogos, sistema e o que mais desejar.

Se está iniciando seus estudos em programação, não tenha dúvida que sua melhor escolha é a linguagem de programação Python.



Python - Vantagens

- Legibilidade** - os programas feitos em Python são muito fáceis de serem 'lidos', você não precisa ficar escrevendo dezenas de linhas de comandos para mostrar apenas um texto na tela (como Java). É como se alguém tivesse conversando com o computador 'Olha, pega esses dados e faz isso...agora joga pra lá, compara com aquilo, organiza e entrega dessa maneira o resultado'.

- Produtividade** - No Python, você não precisa ficar preocupado com memória, alocação de recursos, definição disso e daquilo, ele já faz tudo por você, 'por trás dos panos', quem programa em C por exemplo, precisa ter essas preocupações.

Também não precisa se estressar com sintaxe, ponto e vírgula, etc, pois o código Python é o mais enxuto e minimalista possível.

Assim, o programador só precisa se preocupar com a lógica do programa, nada mais. Resumindo: você escreve menos, e consegue fazer mais que as outras linguagens.

- Portabilidade** - Na gigantesca maioria das vezes, é possível rodar um script em Python tanto no Linux como no Windows ou Mac, sem problema algum, pois a linguagem é altamente portátil. A não ser quando mexe com algo específico do sistema operacional.

- Bibliotecas** - Biblioteca é um conjunto de código com um propósito específico, pra você usar, já pronta. Por exemplo, se quer trabalhar com imagens, vídeo e sons, tem bibliotecas em Python prontas pra isso,

basicamente é pegar e usar, não precisa fazer códigos, boa parte já existe pronto.

Quer trabalhar com ciências, fazer gráficos, simulações 3D, reconhecimento fácil? Tem biblioteca pronta pra isso, alguém já criou, muitos testaram, daí é só usar a biblioteca, suas funções e ser feliz, não precisa inventar o que já inventado

•**Comunidade** - Não importa o que você queira fazer, certamente alguém já fez algo parecido, então pra que começar do 0? Usa o que os outros já fizeram. A comunidade Python é muito, muito grande, e muito, muito unida. Quer fazer um jogo? Provavelmente a parte de som, imagem, lógica do game etc, alguém já fez parecido, e você pode usar.

Já sei programar, devo aprender Python?

Não importa quantas linguagens ou anos de experiência você tem, quando começar a programar em Python vai perceber uma coisa que duvido muito que sentia em outras linguagens: **prazer**.

É bom, é massa, é foda pra caramba programar em Python.
É uma coisa tão simples, tão óbvia, tão enxuta, sucinta e...funciona.

Os programas ficam pequenos, diretos e poderosos.

Coisas que você levaria centenas ou milhares de linhas pra fazer em outra linguagem (como C++ ou Java), você faz com algumas dezenas de linhas de código em Python.

Sabe aquele estresse que dá, definindo tipos, alocando memória, esquecendo ponto e vírgula, escrevendo um monte de coisa do sistema, coisa da linguagem...isso não existe ou é bem minimizado no Python.

É uma linguagem ótima pra resolver problemas rápidos.
Como linguagem para iniciantes, é de longe, a melhor e mais recomendável.

Mas chega de papo, vamos botar a mão na massa.

Baixar, Instalar e Rodar o Python

A grande dúvida de quem nunca programou e quer começar com Python é:

"Onde programo?"

"Onde digito os comandos?"

"Como rodar os programas?"

"Precisa de compilador e fazer coisas complexas?"

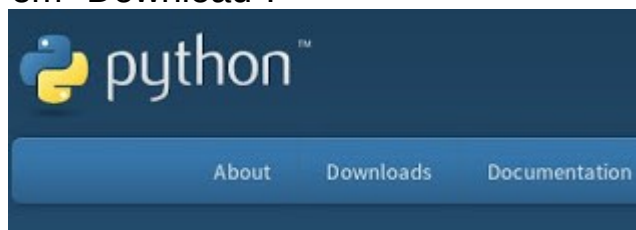
Mas, calma.

Você vai ver como tudo no Python é simples, rápido e fácil, sem enrolação e sem problema.

Primeiro, acesse o site oficial:

<https://www.python.org/>

Em seguida, clique em "Download":



Vai aparecer duas opções pra baixar.

Faça o download da esquerda:



No momento que estou escrevendo este tutorial para nosso **Curso de Python**, a versão mais recente é a 3.6.4 como podem ver na figura.

Quando você for seguir esses passos, certamente a versão já será outra. Baixa ela, sem problemas.

Depois é só seguir o bom velho "Ok", "Avançar", "Aceito os termos...", e prontinho, pra baixar o Python é só isso.

- Interpretador Python

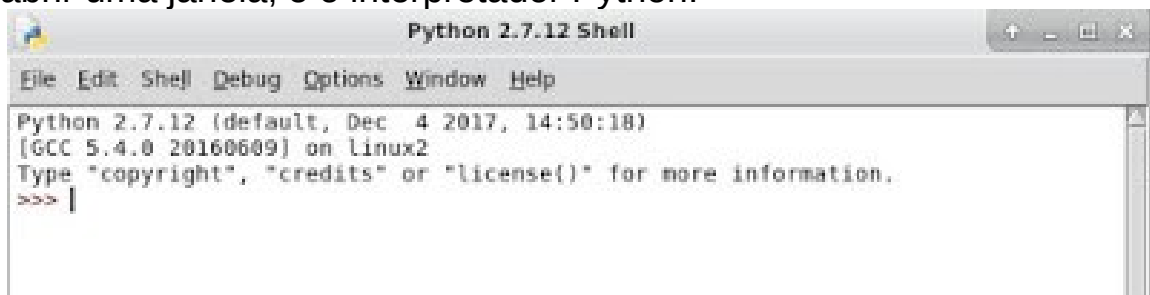
Para você escrever um código Python e fazer ele rodar (funcionar), vai precisar de um interpretador Python. Como nome sugere, ele vai interpretar aquilo que você escreveu e vai executando.

O nome programa que vamos digitar os comandos da linguagem é IDLE (integrated development environment for Python).

Abra o menu iniciar e procure pelo IDLE:



Vai abrir uma janela, é o interpretador Python:



Existem duas maneiras de usar o interpretador.

A primeira, é digitando os comandos nessa janelinha que abriu.

Vamos fazer isso?

Digite 1+1, vai acontecer isso:

```
>>> 1+1
2
>>> |
```

"Olha, uma calculadora, que legal", pode pensar o leitor.

De certo modo, sim, Python é uma calculadora também.

Na verdade, o que ele fez aí foi interpretar, e no entendimento dele você queria o resultado da soma.

Vamos fazer outro teste? Vamos perguntar pra ele se 1 é maior que 2.

Digite 1>2

Como ele interpretou? Essa só vai saber quem experimentou e seguiu todos os passos desse tutorial.

A outra maneira de usar o interpretador, é escrevendo os comandos em um arquivo. Um arquivo de texto mesmo, como o bloco de notas.

O interpretador vai ler esse arquivo do começo ao fim, e linha por linha ele vai interpretando e executando os comandos.

No próximo (e durante o resto do curso de Python) tutorial, iremos mostrar como fazer isso, escrever comandos em um arquivo e colocar ele pra rodar.

Ou seja, vamos começar a criar e rodar scripts escritos em Python!

Conceitos Básicos

Bem vindos a seção Básica, do **Curso Python Progressivo**, a primeira parte dos tutoriais do site.

Nesta parte inicial, vamos aprender o que existe de mais simples e introdutório da linguagem de programação Python.

Vamos supor que você não tenha absolutamente nenhum estudo ou conhecimento em Python ou qualquer outra linguagem. Mesmo que você saiba apenas ligar seu computador, isso já basta para começar a estudar essa seção do site e começar a entender o Python.

Você vai aprender o que é o Python, para que serve, como funciona, o que precisa instalar e digitar para começar a criar *scripts* em Python.

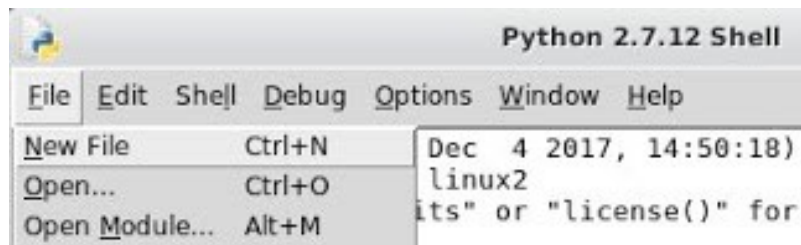
Também iremos te mostrar como exibir mensagens na tela, pegar dados do usuário, fazer cálculos matemáticos e outras coisas bem simples, porém bem interessantes!

Bom estudos!

Hello, World ! - Primeiro Programa em Python

Primeiro de tudo, abra o IDLE.

Provavelmente, vai abrir apenas o interpretador de comandos do Python. Clique em "File" (arquivo, em inglês) e depois em "New File" (novo arquivo), ou simplesmente dê um Ctrl+N:



Note que vai abrir uma nova janela, toda em branco, literalmente parecido com um bloco de notas da vida.

É aí que você vai digitar seu código, onde a mágica vai ocorrer e onde o mundo vai tomar conhecimento do melhor programador da história: você.

Antes de mais nada, crie uma pasta chamada Python, de preferência em um lugar fácil, como no C:\ do Windows ou /home do Linux, para guardar todos seus scripts(programas), de forma bonitinha e organizada

Agora, como bom profissional que é, antes de começar a digitar, você vai adquirir o seguinte hábito:

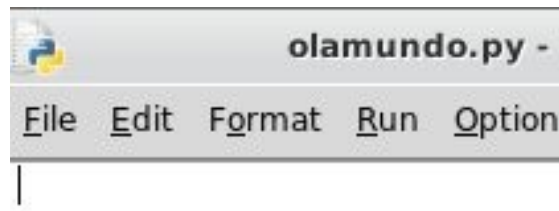
Nesta janela de digitar o código, clique em "File" depois em "Save", ou aperte Ctrl+S, para salvar o programa que vai criar

Em seguida, e **mais importante**: salve seu arquivo com a extensão **.py**
Para o nosso primeiro programa, salve como "olamundo.py"

Arquivos .py são chamados de módulos, estudaremos bastante sobre módulos futuramente.

Tente usar nomes com lógica para seus scripts. Quando ver o 'olamundo.py', sabe que é um programa que faz o Olá, mundo - em Python.

Sua janela de código deve ficar assim:

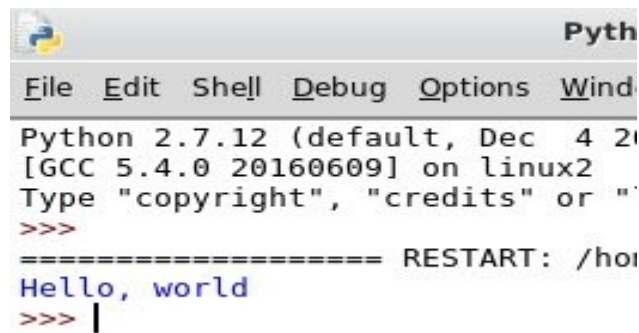


Tudo pronto? Hora da maldade.

Digite naquele 'bloco de notas', a seguinte linha:

```
print 'Hello, world'
```

Agora, vá em "Run -> Run module" ou aperte F5 e veja o que apareceu na janela do interpretador:



Prontinho! Você digitou e rodou o código de um lado, e o resultado dele apareceu na tela do interpretador.

Você acabou de programar um módulo (programa ou script) que exibe uma mensagem de texto ("Hello, world") na tela.

Se estiver usando o Python 3, você pode acabar recebendo uma mensagem de erro, pois nessa nova versão do Python a função print funciona com parêntesis, teste assim:

```
print('Hello, world')
```

Entendendo o primeiro programa em **Python**

Ok, agora vamos entender melhor o que aconteceu ali.

A primeira coisa que você digitou foi **print**, é um comando cuja função é imprimir algo. Calma, não vai sair uma folha de sua impressora.

Imprimir significa mostrar algo na sua tela.

No caso, mostramos um texto, que está entre aspas simples.

Agora digite e rode:


```
print "Olá, mundo"
```

O resultado foi a exibição da frase "Olá, mundo".
O que mudou?

Antes, você usou aspas simples, agora usou aspas duplas.
Ou seja, quando você digita algo entre aspas simples ou duplas, após o comando **print**, tudo que está entre essas aspas é exibido na tela.

Simples e fácil né?
Esse é o Python.

Antes de ir para o próximo tutorial de nosso **Curso de Python**, medite alguns instantes sobre o fato de você ser, oficialmente, um programador, afinal, já criou e rodou seu primeiro programa.

Você é bichão mesmo, hein ?

Função print – Imprimindo coisas na tela

Exibir coisas na tela é, sem dúvida, a comunicação mais básica e uma das mais importantes. Afinal, você não sabe o que um programa está fazendo por trás dos panos, você não vê bits se movendo, nem a memória ou o processador trabalhando.

Somos humanos, precisamos de comunicação humana com nossos scripts Python, e isso é feito através da impressão de informações na tela, e vamos ver agora em como fazer isso através da função **print**.

- **Exibindo coisas na tela em Python**

No primeiro programa que criamos, mostramos como fazer o famoso *Hello, world!* em Python, que é o programa mais simples que podemos fazer.

Agora vamos aprofundar mais nisso, especificamente na função **print**, do Python.

Agora, ao invés de digitar somente:

```
print('Olá, mundo')
```

Experimente escrever e rodar o seguinte código:

```
print('Python')  
print('Progressivo')
```

O resultado será o seguinte:

```
Python  
Progressivo  
>>> |
```

O Python começou a ler seu código do começo pro fim (é um Einstein!), ou seja, de cima pra baixo, da primeira pra última linha de código e foi executando cada comando do seu script.

O primeiro foi imprimir a palavra **Python**, o segundo comando foi imprimir a palavra **Progressivo**. Assim como no código, o resultado foi uma palavra em cada linha.

Exercício: Faça um script que exiba a seguinte mensagem na tela:

```
Curso  
Python  
Progressivo  
>>> |
```

- **Aspas simples e duplas em Python**

Agora, ao invés de aspas simples, vamos usar dupla:

```
print("Python")  
print("Progressivo")
```

O resultado é...tcharam, o mesmo!

Ou seja, tudo que você exibir entre aspas simples ou dupla, o Python interpreta como um texto que vai aparecer na sua tela.

Mais especificamente, ele exibe uma **string**, que nada mais é que uma sequência de caracteres. Estudaremos, mais adiante em nosso curso, com mais detalhes, as danadas das **strings**.

Agora experimente e rode o seguinte código:

```
print('Python')  
print("Progressivo")
```

O primeiro comando está ok.

Já o segundo, começa com aspas simples e termina com aspas duplas...**erro!**

Vai aparecer uma mensagem dizendo que houve um erro de sintaxe! Não pode! Começou com aspas simples? Termine a string colocando aspas simples.

Começou com aspas duplas? Termine com aspas duplas! Tá pensando o que? Python não é bagunça.

- **Imprimindo aspas como string**

Veja que o Python não imprimiu, não mostrou na tela, as aspas, e sim o que está dentro delas.

Joãozinho perguntador: Mas e se eu quiser mostrar as aspas? As vezes vejo programas que mostram, e se eu precisar?

Agora que vem o pulo do gato, você vai ver como o Python é inteligente. Programe e rode o seguinte código:

```
print('Curso "Python" Progressivo')
```

Viu o resultado? Sim, o Python mostrou as aspas duplas:

```
Curso "Python" Progressivo  
>>> |
```

Ele entendeu o seguinte:

a primeira aspa que aparece, é uma simples e a última também. Logo, tudo que tá dentro é uma string, então vou imprimir tudo que tá dentro, do jeito que estiver...to nem aí se tem aspas duplas dentro, elas vão aparecer!

Python, seu lindo! Isso faz sentido!

Exercício: Agora faça o contrário, crie um script que exiba a seguinte mensagem na tela:

```
Curso 'Python' Progressivo  
>>> |
```

Mas ATENÇÃO: você só pode escrever a função **print** UMA VEZ, seu script deve ter apenas UMA linha de código!

Exercícios em Python com a função print

1. Frase na tela - Implemente um programa que escreve na tela a frase "O primeiro programa a gente nunca esquece!".

2. Etiqueta - Elabore um programa que escreve seu nome completo na primeira linha, seu endereço na segunda, e o CEP e telefone na terceira.

3. Letra de música - Faça um programa que mostre na tela uma letra de música que você gosta (proibido letras do Justin Bieber).

4. Mensagem - Escreva uma mensagem para uma pessoa de quem goste. Implemente um programa que imprima essa mensagem, tire um print e mande pra essa pessoa. Diga que foi um vírus que algum hacker instalou em seu computador.

5. Ao site - Faça um programa que mostre na tela o que você deseja fazer usando seus conhecimentos de Python.

6. Quadrado - Escrever um programa que mostre a seguinte figura:

```
XXXXX
X  X
X  X
X  X
XXXXX
```

7. Tabela de notas- Você foi contratado por uma escola pra fazer o sistema de boletim dos alunos. Como primeiro passo, escreva um programa que produza a seguinte saída:

ALUNO(A)	NOTA
=====	=====
ALINE	9.0
MÁRIO	DEZ
SÉRGIO	4.5
SHIRLEY	7.0

8. Letra grande- Elabore um programa para produzir na tela a letra P, de Python Progressivo. Se fosse 'L', seria assim:

```
L
L
L
LLLLL
```

9. Menu - Elabore um programa que mostre o seguinte menu na tela:

Cadastro de Clientes

0 -Fim

1 -Inclui

2 -Altera

3 -Exclui

4 -Consulta

Opção:

10. Pinheiro- Implemente um programa que desenhe um "pinheiro" na tela, similar ao abaixo.

Enriqueça o desenho com outros caracteres, simulando enfeites.

```

      X
     XXX
    XXXXX
   XXXXXXX
  XXXXXXXXX
 XXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXXX
      XX
       XX
      XXXX

```

Tipos de dados em Python - Números, Strings e Booleanos

Veja esta página que está lendo. É um amontoado de texto. Texto, especificamente, cada caractere, é um tipo de dado, um tipo de informação.

Lá nos sistemas da NASA ou de um banco, tem uma porção de números: número da conta, quanto cada cliente tem, quanto pediu de empréstimo, salário dos funcionários...ou seja, temos aí outro tipo de dado: números.

A computação serve basicamente para isso: trabalhar com dados. Basicamente o que os computadores fazem é isso: mexer com dados, informações, manipular, procurar, identificar, exibir isso, calcular aquilo...e é isso que iremos aprender agora.

Nessa parte, não vamos espionar a conta bancária do Trump, invadir os sistemas da NASA nem atrapalhar os planos da Al-Qaeda, como faremos em outros tutoriais. Vai ser um pouco mais teórica, mas de absoluta importância para você se tornar um programador Python.

- **Números em Python**

O tipo mais básico e importante de dado é, sem dúvida, os números. Se o universo fosse um livro, certamente ele seria escrito com números e linguagem utilizada seria a Matemática.

É até desnecessário falar aqui da importância dos números, mas pode ter certeza que existem satélites orbitando por aí pelo espaço, sendo controlado por softwares, que processam dados numéricos, inclusive usando Python.

- **Números inteiros**

Números inteiros são aqueles que não tem parte decimal, como 0, os positivos (1, 2, 3, 4, 5, ..., 2112...) e os negativos (-1, -2, -3, ..., -2112, ...).

Se você usar só números inteiros no Python, ele vai te fornecer números inteiros como resultado:

```
>>> 1+1
2
>>> 3-5
-2
>>> 21 * 12
252
>>> |
```

Dentro destes, tem aqueles chamados inteiros longos, para representar valores realmente estratosféricos como 1111111111111111111L (tem o L no final, de *large*). Dependendo do que quiser fazer, pode ser necessário usar inteiros longos, como para descobrir números primos novos (muito importantes em criptografia de dados).

Calcule: 1111111111111111111111111L + 2222222222222222222222222L, no interpretador do Python.

▪ Números flutuantes

São números que voam, flutuam...brincadeira, nada a ver. É assim que são chamado os números decimais, vulgo 'números quebrados'.

ATENÇÃO ABSOLUTA E MÁXIMA

Em português, usamos a vírgula para representar números decimais: R\$ 1,99 mas em outros países é o ponto. E é o ponto que se usa em programação. U\$ 1.99

Assim, os números flutuantes são escritos assim:

```
0.5
1.5
21.12
```

Se ao menos um de seus dados usados for decimal, o Python já te dá o resultado decimal. Se usar vírgula ao invés de ponto, vai dar problema ou coisas nada a ver:

```
>>> 1 + 1.1
2.1
>>> 1.1 + 1.1
2.2
>>> 21,12 + 21,12
(21, 33, 12)
>>> |
```

- Outras bases decimais

Estamos falando, até o momento, de números na base decimal, que a humanidade costuma usar.

Mas há outros tipos de bases decimais, como a binária, octal e hexadecimal. Os números binários começam com '0b' no início, octal com '0o' (zero e letra o) e os hexadecimais com '0x' no início.

O bacana é que se você digitar um binário, octal ou hexadecimal no interpretador, ele já converte automaticamente pro decimal, veja:

```
>>> 0b10
2
>>> 0o21
17
>>> 0x12
18
.
```

Veja que o número 10 no sistema binário equivale ao valor 2 em nosso sistema decimal.

Agora você entende e está liberado para fazer a seguinte piadinha: existem 10 tipos de pessoas, as que entendem código binário e as que não entendem.



Já se quiser saber quanto vale um número decimal em binário, digite **bin(x)**, onde x é o número que deseja saber. Para octal, **oct(x)** e hexadecimal **hex(x)**.

Exercício: Escreva sua idade no sistema binário, octal e hexadecimal.

- Números complexos

Se já fez segundo grau, certamente já ouviu falar nos números complexos (números fora do plano real, possuem parte real e parte imaginária).

A primeira parte é a real, e a que está com o **j** (número imaginário), é a imaginária.

Calcule a soma de dois números complexos no interpretador do Python:
(1+2j) + (3 + 4j)

Qual foi o resultado?

Agora você pode dizer que está estudando um assunto complexo em programação.

- String – Texto **Python**

Outro importante tipo de dado, são as *strings*, que nada mais são que um conjunto de caracteres.

Seu nome é uma string, seu endereço também, o endereço de um site é uma string, tratamos o IP de um usuário como uma string. Ou seja, todo texto ou símbolo, pode ser tratado como uma string.

Um único caractere, por exemplo, como 'a' ou 'b' é uma string.
1 é um número, mas '1' é uma string.

Somar 1+1 no Python, é ok, afinal, são apenas dois números.
Agora experimente fazer 1 + '1' (lembre-se: strings são representadas entre aspas simples ou duplas), o resultado vai ser um erro:

```
>>> 1+1
2
>>> 1+'1'

Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    1+'1'
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> |
```

Ele alerta que não é possível somar um inteiro com uma string.
Ora, isso faz sentido, não é mesmo? É como somar uma banana com maçã.

Iremos dedicar toooda uma seção de nosso curso para estudar especialmente as strings, afinal, texto é algo de suma importância.

Você pode pegar, por exemplo, o código HTML de uma página, é uma string gigante e tentar achar lá vídeos, fotos, áudio e criar um programa em Python

que faz download de mídias. Para isso, vai ter que 'tratar' a string, achando as coisas que interessa.

É algo bem, mas bem interessante mesmo, trabalhar com strings, dá pra fazer um 'estrago', um programador que sabe lidar bem com elas.

Obviamente, você vai aprender absolutamente tudo, em nosso Curso Python Progressivo, não estranhe se começar a receber propostas de trabalho para NASA, Casa Branca, Estação Espacial, ser um hacker do governo Chinês...

- Booleanos - True e False

Certamente você já ouviu falar que tudo em computação é um amontoado de 0 e 1.

Sabe os filmes que gosta de assistir? Tudo uma combinação de 0 e 1.
Uma foto sua fingindo estar meditando na natureza? Tudo 0 e 1.
Sua música favorita? 0 e 1.

Na verdade, um computador não tem a menor ideia do que são números, strings ou nada disso, ele só entende 1 e 0, que na verdade é uma interpretação das voltagens (alta chamamos de 1, sem voltagem é 0).

Pois bem, como somos programadores Python profissionais, vamos nos aprofundar nisso.

Existe um tipo de dado em Python chamado **booleano**, ele é bem simples, só pode assumir dois valores: **True** (verdadeiro) e **False** (dã, adivinha).

Pronto. Só isso.

Se convencionamos que 1 é verdadeiro, True.

E 0 é falso, False.

Um programador não fala 'falsiane' e sim 'zeriane', xingamos dizendo 'você é muito 0, não confio em você'.

Já se quiser elogiar alguém, diga 'você é muito 1, por isso me apaixonei'.

Vamos ver como o Python interpreta esse tipo de dado.

1 é maior 2? E 3 é maior que 2 ?

```
>>> 1>2
False
>>> 3>2
True
>>> |
```

Python é o(a) namorado(a) ideal, tudo que você pergunta ele responde e com sinceridade, sempre falando a verdade.

Exercício: descubra porque chamamos de 'boolean', esse tipo de dado. Sim, um bom programador é aquele capaz de pesquisar, fuçar e descobrir as coisas por si só.

- Outros tipos em **Python**

Na verdade verdadeira, sendo bem rigoroso, os dados em Python são objetos.

Você vai entender melhor isso quando estudarmos orientação à objetos.

Quando falamos em números, strings, caracteres, booleanos...falamos de alguns tipos de dados, os principais e mais utilizados.

Porém, há muitos outros, que inclusive iremos usar bastante (mesmo), mais a frente em nosso **curso de Python**, como:

- Arquivos
- Tuplas
- Listas
- Dicionários

Todos eles são dados básicos e internos.

São internos, porque já são incorporados no Python.

O Python é tão foda, mas tão foda, que vamos poder até mesmo criar nosso próprio tipo de dado, nossa própria estrutura de informação.

Se você quiser que 'queijo' seja um tipo de dado, como número é um dado, a letra 'A' é um dado, então ele vai ser dado, e vai ser como você quiser, comportar como você quiser e ponto final.

Se quiser que a raiz quadrada de um queijo seja leite, vai ser, pois você define o que é seu dado e como ele se comporta.

Mas por hora, vamos deixar isso apenas como curiosidade.
Sigamos nosso curso...

Variáveis em Python

No tutorial anterior, falamos sobre os **tipos de dados**, onde enfatizamos o estudos dos números, strings e booleanos.

Agora vamos ver onde e como armazenamos esses dados, estudando as variáveis em Python.

- Armazenando dados na memória

Quando você liga seu computador e abre seu e-mail ou Facebook, provavelmente ele já faz o login diretamente. Mas, como eles sabem qual sua conta? Seu e-mail e senha?

Simples: estas informações estavam guardadas em algum lugar.

Tem algum aplicativo de música no celular ou rádio no carro?
Se você desligar um dia e tiver em uma música do Rush, nos 21min e 12s, quando ligar novamente, vai abrir naquela música, naquele tempo da música.

Mágica? Bruxaria? Sorte?

Óbvio que não, aquela informação foi guardada em algum lugar.

E o carrinho de compras dos sites?

Você entra, escolhe uns livros de Python pra comprar...desiste de comprar.
Quando voltar no site, vai estar lá ainda as opções que queria.

Como isso acontece?

Certamente isso ficou guardado em algum lugar, essa informação.

Concorda ?

É aí que entram as variáveis.

- Variável em **Python**

A maneira na qual vamos armazenar informações, dados, através da programação Python, é usando variáveis.

Variável nada mais é que um nome que vamos dar a um determinado bloco de memória. Quando o computador quer salvar, por exemplo, o número de IP de um usuário no bloco de memória 0xH2112 , é muito ruim para um humano (programador), ter que ficar usando esses números, são difíceis de decorar e manipular.

Ao invés disso, criamos uma variável, por exemplo, de nome '**ip_usuario**', e pronto, sempre que quisermos usar esse dado, usamos a variável '**ip_usuario**', ao invés de ter que usar diretamente o bloco de memória onde esses dados estão armazenados.

Um pouco teórico, não é? Vamos deixar de papo e partir pra prática.

- **Como usar variáveis em Python**

Para criarmos uma variável em Python, temos simplesmente que fazer uma declaração de atribuição.

Vamos criar, por exemplo, uma variável que vai armazenar um número, sua idade, por exemplo. O nome dela vai ser '**idade**' (uau, que original).

Vamos declarar essa variável (dizer ao Python: 'ei, se liga, é uma variável, ok?') e vamos atribuir a ela um valor, no caso, um número inteiro positivo.

Se você tem 18 anos e quer atribuir esse valor a uma variável chamada '**idade**', basta fazer:

- `idade = 18`

Sim, só isso. O Python vai entender que '**idade**', é uma variável, é algo seu, que você, programador, criou. Vai alocar (reservar) um espaço na memória (um espaço localizado no endereço 0xFFFF4h , por exemplo), e lá vai guardar o valor 18.

Sempre que quiser imprimir essa idade, usar em um programa pra saber se pode dirigir, se pode votar etc etc, não precisa decorar o endereço 0xFFFF4h de memória, apenas use a variável '**idade**', ela automaticamente vai ser uma referência para aquele endereço de memória.

- **Imprimindo variáveis na tela**

Vamos criar uma variável que, dessa vez, armazenar uma string. Essa variável vai se chamar '**texto**', e vamos colocar lá a string 'Curso Python Progressivo'.

Basta fazermos isso em nosso código Python:

- `texto='Curso Python Progressivo'`

Se escrever essa linha e rodar, parece que não acontece nada. Mas

acontece: o Python vai armazenar um espaço de memória e lá vai guardar o texto **Curso Python Progressivo**, depois vai encerrar o programa, pois é só isso que seu código faz.

Não é porque não apareceu nada na tela que nada ocorreu. Ocorreu, mas por trás dos panos, ok?

Mas como dissemos antes, a função **print** serve para imprimir (exibir) na tela alguma coisa (texto, número, booleano etc). Mas a variável 'texto' armazena uma string...

...então, para imprimir ela, basta programar o seguinte código:

```
texto='Curso Python Progressivo'
print texto
```

Veja como fica o código (janela de cima) e o resultado no interpretador Python (janela de baixo):



- **Imprimindo mais de uma variável**

A função **print** do Python é bem mais poderosa e versátil do que aquilo que já estudamos.

Podemos, por exemplo, imprimir mais de um dado no mesmo comando, basta separarmos por vírgula.

O programa a seguir armazena sua idade em uma variável e seu nome em outra. Em seguida, imprime tudo numa mesma printada violenta:

```
nome='Maria Joaquina de Amaral Pereira Goes'
idade=18
print nome,idade
```

Faça e veja o resultado.

Note que a variável **idade** armazena um número, e não uma string, porém, a função print mostra ela do mesmo jeito.

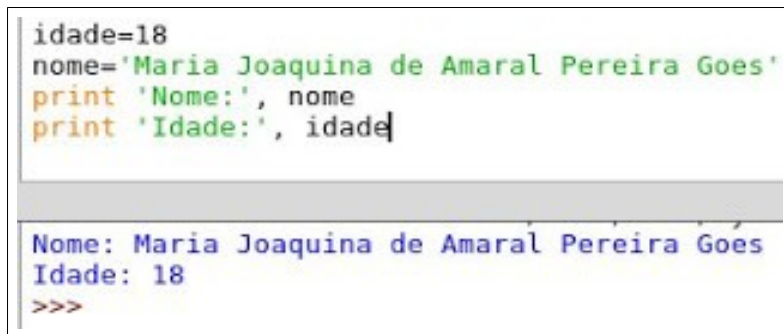
Porém, saiu tudo na mesma linha, um nome e um número.

Ficou um pouco feio, vamos embelezar um pouco.

Digite e rode o seguinte código:

```
idade=18
nome='Maria Joaquina de Amaral Pereira Goes'
print('Nome:', nome)
print('Idade:', idade)
```

Aaaaah! Agora sim! Fala sério, ficou bem bonitinho e bem arrumadinho o resultado, vejam:



```
idade=18
nome='Maria Joaquina de Amaral Pereira Goes'
print 'Nome:', nome
print 'Idade:', idade

Nome: Maria Joaquina de Amaral Pereira Goes
Idade: 18
>>>
```

- **Exercícios usando variáveis em Python**

Exercício 01:

Crie um programa que exiba na tela seu nome completo, sua cidade, estado e data de nascimento.

Para cada dado desse, crie uma variável apropriada. Use nomes que façam sentido ('cidade', 'estado' etc, nada de ficar criando variáveis com nomes 'a', 'b', 'x' ou 'y' - isso é um péssimo hábito entre programadores).

Exiba tudo na tela, bonitinho e organizado.

Exercício 02:

Crie um programa que exiba na tela o texto 'A melhor banda do mundo é [nome da banda] e a melhor música é [nome da música]'.

O nome da banda e o nome da música devem estar declarados em duas

variáveis diferentes.

A saída deve estar assim:

```
A melhor banda do mundo é Rush e a melhor música é 2112  
>>>
```

PS: Se estiver algum problema para rodar o código, pode ser problema para exibir alguns caracteres como 'é' e 'ú' de música.

Para resolver esse problema, adicione o seguinte código no início de seu script:

```
# encoding: utf-8
```

- Mudando valores de variáveis

Sabe por que se chama variável?
Porque varia.



Aliás, é extremamente normal as variáveis mudarem de valor, no decorrer de um programa.

Vamos dar um exemplo agora usando número decimal.

Inicialmente, vamos dar o valor '0.0' para uma variável, em seguida mudar para 5000.00 por exemplo:

```
salario=0.0
```

```
print('Antes de ser programador eu ganhava R$',salario,'por mes')
```

```
salario=5000.0
```

```
print('Agora virei programador Python e ganho R$',salario)
```

O script vai sempre imprimir o valor que está no endereço de memória apontado pela variável '**salario**'.

Inicialmente, lá na memória, tá armazenado o valor 0.0, então o primeiro print imprime 0.0

Em seguida, mudamos o valor lá da memória pra 5000.0

O valor 0.0 já era, mudou, se perdeu, agora tem gravado cinco mil lá no bloco de memória, e como a variável '**salario**' aponta pra esse endereço, vai imprimir o que tiver lá, e agora imprime 5000.0

Perfeitamente lógico e simples, não é ?

Isso se chama reatribuir valor a uma variável.

- **Regras para declarar uma variável**

Você não sair escolhendo qualquer nome para uma variável.

Algumas palavras são ditas 'reservadas', pois são de uso da linguagem, que são:

```
['and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'exec', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'not', 'or', 'pass', 'print', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Outras regras para declarar uma variável:

- Deve começar com letra ou *underscore* (underline)
- Após o primeiro caractere, pode usar dígitos
- Maiúsculo é diferente de minúsculo: a variável '**python_progressivo**' é diferente da variável '**Python_Progressivo**', mudou um caractere qualquer entre maiúsculo e minúsculo, muda tudo. '**teste**' é uma coisa e '**Teste**' é outra, cuidado com isso!

Outra dica importante, que não é regra, mas mostra que você é um bom programador Python, é usar variáveis que façam sentido.

Se você for contratado para fazer o sistema do Detran, use variáveis de nome 'carro', 'velocidade', 'cnh', 'validade' ao invés de 'a', 'b', 'c'...pois só de passar os olhos no nome das primeiras, você entende que tipo de informação ela está armazenando.

No começo, isso pode parecer inútil. Mas a medida que for criando

programas maiores e mais complexos, de centenas de linhas de códigos ou mais, isso é essencial, é uma chamada **boa prática de programação**.

É bem comum nomes de variáveis terem mais de uma palavra:
pythonprogressivo

Porém, as vezes é ruim de ler. Algumas coisas facilitam, como:
python_progressivo

Ou *ocamelCase*, que é usar a primeira letra da primeira palavra minúscula, e a primeira letra das próximas palavras maiúsculas:

cursoPythonProgressivo

Facilita ler, não é verdade?

Exercício: O seguinte código imprime todas as palavras reservadas (*keywords*) do Python, rode ele e veja o resultado:

```
import keyword  
print(keyword.kwlist)
```

A função input() do Python - Lendo dados do Teclado

Até o momento, em nosso **Curso de Python**, nossos scripts não tem interação nenhuma entre o Python e o usuário.

Eles simplesmente rodam do começo ao fim, sempre do mesmo jeito.

Mas não é isso que ocorre corriqueiramente nos programas que usamos.

Fornecemos dados (como textos, login, senhas), clicamos em coisas, recebemos dados da internet, até um ato de apertar um botão do PC é uma comunicação do usuário com a máquina.

Neste tutorial, vamos te ensinar como começar a receber dados das pessoas que estão executando os programas, através da função **input()** do Python!

- Como Receber Dados – Função **input()** do **Python**

O formato da função input é o seguinte:

variavel = input(string)

Só isso.

Seja lá o que a pessoa digitar, a informação ficará armazenada na variável de nome 'variavel'. E o que será exibido na tela é a string (texto) 'string'.

Vamos ver na prática o uso da função **input**. Programe o seguinte código:

- `variavel=input('Digite algo: ')`

O resultado dele vai ser:

```
Digite algo: 2112
>>>
```

Prontinho.

Como fornecemos a string 'Digite algo: ' para a função **input**, foi exatamente isso que foi exibido na tela.

Em seguida, o interpretador Python simplesmente fica parado, esperando você digitar algo. Enquanto você não apertar enter, nada vai acontecer.

Quando você pressiona enter, ele continua.

No caso, nosso script serve apenas para armazenar o que digitamos na variável '**variavel**', na forma de uma string.

Atenção: a função input armazena na forma **string** se tiver usando a versão recente do Python de número 3.x ok?

Se for versão antiga, ela vai transformar seu dado em string, inteiro ou float, dependendo do que você digitar.

Atualize seu Python! Use a versão mais nova!

- Exercício com a função **input()** do Python

"Faça um programa que pergunte a idade do usuário, e a armazene em uma variável. Em seguida, pergunte o nome da pessoa e armazene esse dado em outra variável. Por fim, exiba uma mensagem de boas vindas ao curso Python Progressivo, dizendo nome e idade da pessoa".

Solução comentada

Inicialmente, vamos armazenar a idade do usuário na variável 'idade', e usamos a função **input** para receber tal dado.

Depois, faremos o mesmo com o nome, armazenando na variável 'nome'.

Por fim, damos um print onde escrevemos uma mensagem de saudação e imprimimos também o nome e a idade da pessoa, que estão armazenadas nas variáveis 'nome' e 'idade', veja como ficou simples:

```
idade=input('Qual sua idade? ')
nome=input('Qual seu nome? ')
```

```
print('Olá, seu nome é ', nome, ' e tem ', idade, ' anos de idade! Seja bem vindo ao Curso Python Progressivo')
```

O resultado é o seguinte:

```
===== RESTART: /home/user/input.py =====
Qual sua idade? 18
Qual seu nome? "Bruce Dickinson"
Olá, seu nome é "Bruce Dickinson" e tem 18 anos de idade! Seja bem vindo ao
Curso Python Progressivo
>>> |
```

Note que digitamos o nome "Bruce Dickinson" entre aspas, isso é necessário

se vocês estiver usando uma versão mais antiga do Python, se não usar, vai receber uma mensagem de erro.

Se estiver na versão mais recente (a que estou usando, no momento que escrevo este tutorial de nosso curso, é a 3.6), não precisa usar aspas, a função **input** passa tudo pra string.

Se quiser digitar uma string sem precisar de aspas, em versões mais antigas do Python, ao invés de **input** use **raw_input**

Pronto, finalmente você está se comunicando com o Python.

Ele espera, aguarda, fica quietinho e ansiosamente esperando que você forneça as ordens. Você que manda na p..rra toda !
Afinal, você é o programador, também conhecido como 'dono(a) do universo'.

Com grandes poderes, vem grandes responsabilidades.

Em um próximo tutoriais vamos criar uma calculadora em Python. Sinta o poder em suas mãos.

Como transformar strings em números em Python

- Funções int() e float()

No próximo tutorial de Python, vamos começar a trabalhar com Matemática, fazer operações e até criar uma calculadora bem bacana e simples, toda feita com o que estudamos até o momento.

Mas antes de falarmos disso, precisamos resolver um problema com nossa função **input** que recebe dados do teclado do usuário.

- **Função input() e retorno em string**

Quando digitamos qualquer coisa no teclado para a input capturar, ela automaticamente vai transformar esse dado em uma string (um texto).

Atenção: Isso na versão mais nova do Python, a 3.x (3.4, 3.6 ...etc).
ATUALIZE SEU PYTHON PRA VERSÃO MAIS NOVA, OK?

Vamos usar o seguinte script que pede três dados do usuário e em seguida exibe o tipo de **variável** através da função **type()**.

Nosso script:

```
string=input("Digite uma string: ")  
print( type(string) )
```

```
inteiro=input("Digite um inteiro: ")  
print( type(inteiro) )
```

```
decimal=input("Digite um float: ")  
print( type(decimal) )
```

Agora vamos rodar e digitar uma string, depois um inteiro e depois um float (decimal), veja o resultado:

```
Digite uma string: Curso Python Progressivo  
<class 'str'>  
Digite um inteiro: 2112  
<class 'str'>  
Digite um float: 21.12  
<class 'str'>  
>>> |
```

Veja que nos três casos, as variáveis são do tipo string ('str').
Mas, peraí Python!

Eu queria que só a primeira fosse string!
2112 queria que fosse um inteiro
21.12 queria que fosse um decimal (float)

Por que isso acontece? Deu bug no Python? Quebrei o Python?

Não, caro paladino das artes computacionais.
Isso é uma característica da função **input()** no Python 3.x

Dizemos que a função input retorna uma string.
Ou seja, ao utilizarmos ela, ela vai colocar uma string na variável, não importa o que você tenha digitado, ok?

Mas vamos aprender como transformar essas strings indesejadas em números do tipo que quisermos!

- Função **int()** - String para Inteiro (str to int) em **Python**

Para transformar uma string em inteiro, vamos usar a função **int()**
Basta colocarmos o que quisermos entre os parêntesis dessa int(), que ela *retorna* um inteiro.

No caso, vamos colocar uma string, ok?

Exemplos:

Vamos definir uma variável chamada **var1** e colocar a string '2112' dentro dela. Em seguida, vamos usar a função **type()** para exibir o tipo de dado (vai aparecer 'str' de string).

Em seguida, vamos pegar outra variável **var2** e fazer ela receber a função **int()** e dentro dessa int, colocamos a string **var1**. Depois imprimimos o tipo de dado que é a var2, veja:

```
var1='2112'  
print( type(var1) )  
  
var2 = int(var1)
```



```
print( type(var2) )
```

O resultado é exatamente o que esperamos:

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

Transformamos a string '2112' no número 2112 !

- Função **float()**- String para Decimal (str to float) em **Python**

Exatamente como funciona a **int()**, funciona a **float()**.

Tudo que colocarmos entre os parêntesis dessa função, ela vai transformar em float.

Vamos criar um script que pede um decimal para o usuário e armazena na variável **var1**. Depois, imprimimos o tipo de dado dessa var1, que vai ser 'str'.

Depois, colocamos essa var1 dentro da função **float()** e armazenamos o valor em uma variável **var2**. Em seguida, imprimimos o tipo de dado dessa variável, que vai ser 'float'.

Agora, vamos pegar uma variável **var3** e colocar um inteiro nela, o número 2112, que é o número mais foda de todos.

Depois transformamos esse inteiro em um float, armazenamos ele em **var4** e vemos o tipo de dado da var4, que agora agora é float, veja:

```
var1=input("Digite um decimal: ")
print( type(var1) )
```

```
var2=float(var1)
print( type(var2) )
```

```
var3=2112
var4=float(var3)
```

```
print( type(var4) )
```

```
print(var4)
```

O resultado foi esse:

```
Digite um decimal: 21.12
<class 'str'>
<class 'float'>
<class 'float'>
2112.0
>>> |
```

Xablau! Transformamos uma string em float, e depois o inteiro 2112 em um float. Imprimimos até **var4** para vermos que 2112 se transformou em 2112.0 , um decimal!

- **Usando int() e float() na função input()**

Ok, mas e o nosso problema inicial? De querer um número e o diacho da função **input** me dar uma string, como resolver?

Muito simples. Lembra que dissemos que a input *retorna* uma string? Então é só jogar a função input **dentro** das funções int() e float()

Ou seja, para pegar um dado do usuário, transformar em inteiro e armazenar na variável **var1**, faça:

```
var1 = int( input("Digite um inteiro: ") )
```

E para pegar o dado do usuário e transformar em decimal, basta jogar a input dentro da função float:

```
var2 = float( input("Digite um decimal: ") )
```

E prontinho! **Var1** é um tipo de dado inteiro e **var2** é um tipo de dado float. Simples, não?

Operações Matemáticas no Python - Adição (+), Subtração (-), Multiplicação (*), Divisão (/), Exponenciação (**) e Resto da divisão (%)

Computação...já parou pra pensar o que significa essa palavra ?
Vem de computar, que significa calcular.

Sim, basicamente o que um computador fazer é isso: contas. Muitas e bem rápido.

Neste tutorial de nosso **Curso de Python**, vamos aprender a somar, subtrair, multiplicar, dividir, exponenciar e calcular o resto da divisão (que diacho é isso?)

- Como **Somar** em **Python**: +

O operador de soma, em Python, é...adivinha, o símbolo: +
Surpresa, hein?

Vamos fazer um script que pede um número inteiro ao usuário, armazena em **var1**, depois outro inteiro e armazena em **var2**.

Em seguida, fazemos a soma desses dois números e armazenamos na variável **soma**, e printamos a soma. Digite e rode o seguinte código:

```
var1 = int( input("Digite um inteiro: ") )  
var2 = int( input("Digite outro inteiro: ") )
```

```
soma = var1 + var2  
print(soma)
```

Legal, né ?

- Como **Subtrair** em **Python**: -

Se você achou que o símbolo de subtração, em Python, fosse o - , parabéns, você é sério candidato para ganhar o próximo prêmio Nobel.

Vamos criar um script que pede dois números, subtrai um do outro e exibe o resultado:

```
var1 = int( input("Digite um inteiro: ") )  
var2 = int( input("Digite outro inteiro: ") )
```

```
subtracao = var1 - var2
```

```
print(subtracao)
```

Note que você só pode fazer a subtração depois de fornecer os números. Se fizer o 'subtracao = var1 - var2' no começo, vai dar um erro, pois o Python ainda não sabe que valores estão em var1 e var2, pois você ainda não forneceu nada!

- Como **Multiplicar** em **Python**: *

Finalmente algo diferente! Sim, o símbolo de multiplicar não é x, é o asterisco *

```
var1 = int( input("Digite um inteiro: ") )  
var2 = int( input("Digite outro inteiro: ") )
```

```
produto = var1 * var2
```

```
print(produto)
```

Escreva o código acima, rode ele, várias vezes, faça testes, coloque a mão na massa, ok? Só ficar passando o olho aqui não vai te fazer um bom programador Python.

É precisa *codar*, ou seja, digitar os códigos, na mão!

- Como **Dividir** em **Python**: /

Já o símbolo de dividir é o /

Ou seja: $4/2 = 2$

Veja o script que pede dois números ao usuário e exibe a divisão deles:

```
var1 = int( input("Digite um inteiro: ") )  
var2 = int( input("Digite outro inteiro: ") )
```

```
divisao = var1 / var2
```

```
print(divisao)
```

Teste: Na segunda variável, que vai ser o denominador, teste colocar 0. O que aconteceu? Por quê ?

- **Exponenciação em Python: ****

Exponenciar, se você já esqueceu, é o famoso 'elevar' e seu símbolo são dois asteriscos juntos: **

Por exemplos, 3 elevado a 2:

$3 ** 2 = 9$, pois $3 \times 3 = 9$

3 elevado a 3:

$3 ** 3 = 27$, pois $3 \times 3 \times 3 = 27$

Rode o seguinte script:

```
var1 = int( input("Digite um inteiro: ") )  
var2 = int( input("Digite outro inteiro: ") )
```

```
exp = var1 ** var2
```

```
print(exp)
```

Teste: Use números enormes, gigantescos, medonhos.

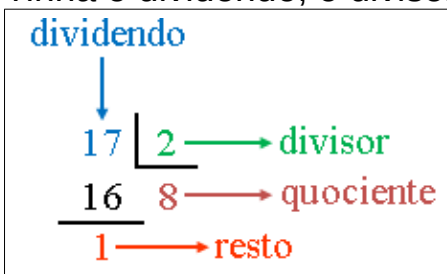
E aí, o Python calculou ? Foi rápido? Fodástico esse Python, não é?

- Resto da Divisão: %

Essa operação talvez você não lembre.

Vamos voltar lá pra escolinha, quando fazíamos as continhas de dividir, lembra?

Tinha o dividendo, o divisor, o quociente e o resto, veja:



Para saber o resto da divisão de um número por outro, usamos o operador %
Veja o resto da divisão de números pares por 2, teste:

```
var1 = int( input("Digite um inteiro: ") )  
var2 = int( input("Digite outro inteiro: ") )
```

```
resto = var1 % var2
```

```
print(resto)
```

Vai dar sempre 0 né?

Agora teste o resto da divisão de um número ímpar por 2.

O resto vai ser sempre um.

Vamos usar o operador de resto da divisão para isso, por exemplos: achar números pares. Vamos usar também para achar números primos também!

É um operador muito importante e útil no mundo da programação, ok?

- Exercício Mais Importante de Python

Ok! Agora, você vai precisar fazer este exercício.

Só continue em nosso curso se resolver ele.

Nem que fique grande, feio ou confuso, **mas faça esse exercício.**

Exercício: Crie um programa em Python que peça dois números ao usuário. Em seguida, você vai mostrar a soma, subtração, multiplicação, divisão, exponenciação e resto da divisão do primeiro número pelo segundo.

Tem que ficar bonitinho e organizadinho assim, o resultado:

```
Digite um numero: 21
Digite outro numero: 12
Soma:                21.0 + 12.0 = 33.0
Subtração:           21.0 - 12.0 = 9.0
Multiplicação:       21.0 * 12.0 = 252.0
Divisão:             21.0 / 12.0 = 1.75
Exponenciação:       21.0 ** 12.0 = 7355827511386641.0
Resto da divisão:    21.0 % 12.0 = 9.0
>>>
```

Exercícios Resolvidos de Porcentagem e Juros Compostos

◦ Exercício 01 de Porcentagem em Python

Ao terminar o **Curso Python Progressivo**, você foi disputado a tapa por várias empresas, e uma grande varejista te ofereceu o salário maior.

Sua primeira tarefa é criar um programa em Python que pede o preço original de um produto e dá 20% de desconto.

Você deve mostrar uma tabela contendo:

- Preço original do produto
- Valor do desconto em R\$ (tipo 'Você ganhou R\$ xx,xx de desconto')
- Valor do produto com o desconto

◦ Exercício 02 de Porcentagem em Python

A loja percebeu que não quer dar 20% em tudo. Quer dar 20% em algumas coisas, 10% em outra, nada em outro produto e até 30% em alguns outros produtos.

Crie um script em Python que pergunte o preço original e o desconto que deve ser concedido.

Ele deve mostrar a tabela igual a do exercício anterior.

Exercício 03 de Juros Compostos em Python

Uma boa parte dos sistemas bancários foram programados há décadas.

O diretor de um grande banco decidiu que era hora de se modernizar, usando uma linguagem mais moderna, segura e poderosa.

Obviamente, escolheu o Python e contratou você para trabalhar no novo sistema.

Sua primeira tarefa é criar um programa envolvendo a poupança.

Você vai perguntar o valor inicial investido na poupança, a rentabilidade mensal, quantos meses o cliente deseja deixar o dinheiro investido e mostrar o valor final na conta do cliente do banco.

- Exercício de juros compostos

Cliente é um bicho folgado. Vive inventando e pedindo coisas, e tudo acaba sobrando para o coitado do programador.

Um cliente pediu que o sistema do banco tivesse a seguinte função: Dizer o valor inicial que ele deve investir, para ao final de 'm' meses ele tenha um valor 'vf', supondo que este dinheiro esteja rendendo uma rentabilidade 'i' mensal, em porcentagem esse 'i'.

Faça um programa que pede o valor final, o tanto de meses que vai ficar aplicado, a rentabilidade e o script mostre o valor inicial que ele deve investir para atingir tal objetivo.

Solução dos exercícios

- Exercício 01 resolvido

Aplicar um desconto de 20% nada mais é que multiplicar o valor por 0,2

Então, inicialmente pegamos o valor original e armazenamos na variável 'valor_original'. Não esqueça de passar a input para a função float, para virar um número decimal.

Em seguida, multiplicamos esse valor original por 0.2 (em inglês se usa ponto e não vírgula, para números decimais) e armazenamos na variável 'valor_descontado', é o desconto que o cliente vai ganhar.

Como a pessoa ganha 20% de desconto, significa que ela vai pagar apenas 80% do valor original. Ou seja, para saber o valor final, é só multiplicar o valor inicial por 0,8

Nosso código fica:

```
# Vamos pedir o valor original do produto
valor_original = float( input("Valor original: R$ ") )
```

```
# Desconto ganho de 20%
valor_descontado = valor_original * 0.2
```

```
# Valor com desconto incluso
novo_valor = valor_original * 0.80
```

```
# Exibindo tudo
print("Valor original:    R$", valor_original)
print("Desconto ganho:    R$", valor_descontado)
print("Valor com desconto: R$", novo_valor)
```


Uma outra maneira mais 'enxuta' de resolver seria:

Vamos pedir o valor original do produto

```
valor_original = float( input("Valor original: R$ ") )
```

Exibindo tudo

```
print('Valor original:   R$', valor_original)
```

```
print('Desconto ganho:   R$', valor_original * 0.2)
```

```
print('Valor com desconto: R$', valor_original * 0.8)
```

Simples, não é?

Até parece bobo, mas toda loja tem um script desse, para esses cálculos.

```
Valor original: R$ 100
Valor original:   R$ 100.0
Desconto ganho:   R$ 20.0
Valor com desconto: R$ 80.0
>>> |
```

- Exercício 02 resolvido

Agora, não sabemos o valor do desconto de cada produto.

Então, além de perguntar o valor original do produto, também temos que perguntar o valor do desconto e armazenar numa variável chamada 'desconto'.

Mas aqui que mora o perigo.

Quem vai fornecer o valor do desconto vai fornecer em porcentagem.

Tipo, vai escrever '10' para 10%. Porém na hora de dar o desconto, a gente multiplica por 0.10 e não por 10.

Então, o que fazer? Simples: pegamos o valor que a pessoa vai digitar e dividimos por 100 (porcentagem = por **cem** tagem).

Aqui temos o pulo do gato.

Temos a variável 'desconto' com valor 10.

Como transformar ela pra ter o valor 0.1 armazenado?

Pode declarar uma nova variável, a **desconto2**, que é:

```
desconto2 = desconto / 100
```

Ou mais comum em programação:

```
desconto = desconto / 100
```

Isso quer dizer o seguinte: "Python, o novo valor de **desconto** é o valor

antigo dela dividido por 100".

O valor que a pessoa ganha de desconto é o valor original multiplicado por essa variável **desconto**.

E o novo valor, incluído o desconto?

Se 10% é o desconto, então o valor com desconto incluso é: valor original multiplicado por 0.9 (de 90% = 100% - 10%)

Basta fazer: $1 - \text{desconto}$

Se o desconto é 0.3 , então vamos multiplicar o valor original por: $1 - 0.3 = 0.7$

Em código Python, fica assim: **valor_original * (1 - desconto)**

O código final fica:

Vamos pedir o valor original do produto

```
valor_original = float( input("Valor original: R$ ") )
```

Desconto que será concedido

```
desconto = float( input("Desconto (em %) para esse produto: ") )
```

Transformando a porcentagem em número decimal

```
desconto = desconto / 100
```

Exibindo tudo

```
print('Valor original:   R$', valor_original)
```

```
print('Desconto ganho:   R$', valor_original * desconto)
```

```
print('Valor com desconto: R$', valor_original * (1-desconto) )
```

E prontinho! Não importa o valor do produto nem do desconto, o que o funcionário da loja digitar, seu programa vai mostrar tudo bem detalhado: valor original, desconto ganho e valor com desconto incluso:

```
Valor original: R$ 200
Desconto (em %) para esse produto: 30
Valor original:   R$ 200.0
Desconto ganho:   R$ 60.0
Valor com desconto: R$ 140.0
>>> |
```

Agora quando você for em um supermercado, farmácia ou qualquer loja que tenha um sistema de preços e descontos, você já sabe como funciona e já sabe criar.

Não se preocupe, no decorrer de nosso **Curso de Python**, vamos aprender a criar aquelas programinhas com janelas, botões e tal.

Mas o principal, a lógica da coisa, como funciona por trás dos panos, você já sabe como é.

Bacana hein? Daqui a pouco seu telefone toca, com lojas querendo te contratar, afinal você é um programador Python que aprendeu a programar no Python Progressivo.

- Exercício 03 resolvido

A fórmula dos juros compostos é a seguinte:

$$V_f = V_i \cdot (1+i)^m$$

Onde:

Vf - Valor final na poupança, ao término do tempo

Vi - Valor inicial que o cliente vai colocar na poupança

i - Rentabilidade mensal (em porcentagem)

m - Tanto de meses que o dinheiro do cliente vai ficar rendendo

O primeiro passo, é perguntar o montante inicial que vai ser investido. Vamos armazenar esse número na variável 'vi'.

Em seguida, perguntamos a rentabilidade mensal, em porcentagem e armazenar na variável 'i'.

Como esse 'i' está em porcentagem, devemos dividir ele por 100 para aplicar na fórmula. Assim, se a rentabilidade for 0,5% , o valor de i será i=0,005

Para isso, basta fazer: $i = i / 100$

Por fim, pergunte o número de meses que a aplicação vai ficar rendendo, e armazene na variável inteira 'm'.

Nossa fórmula, em código de programação Python, é:

```
vf = vi * (1+i)**m
```

Concorda ?

Nosso código fica assim:

```
# Valor inicial aplicado
```

```
vi = float(input('Valor inicial: '))
```

```
# Rentabilidade mensagem, em %  
i = float ( input('Rentabilidade mensal: ') )  
  
# Transformando a porcentagem em valor numérico  
i = i / 100  
  
# Tempo de investimento  
m = int( input('Meses que vai deixar rendendo: ') )  
  
vf = vi * (1+i)**m  
  
print('Valor apos ',m,' meses: R$ ',vf)
```

Vejamos uma simulação:

Bem simples, não é?

Criando uma calculadora simples em Python

No tutorial passado de nossa **Apostila de Python**, aprendemos a somar, subtrair, dividir, multiplicar, exponenciar e achar o resto da divisão, de dois números quaisquer.

Propusemos que você criasse uma calculadora em Python, que pede ao usuário dois números e exiba todas as operações da seguinte maneira:

```
Digite um numero: 21
Digite outro numero: 12
Soma:          21.0 + 12.0 = 33.0
Subtração:     21.0 - 12.0 = 9.0
Multiplicação: 21.0 * 12.0 = 252.0
Divisão:       21.0 / 12.0 = 1.75
Exponenciação: 21.0 ** 12.0 = 7355827511386641.0
Resto da divisão: 21.0 % 12.0 = 9.0
>>>
```

Agora vamos mostrar a solução desse exercício, de duas maneiras. Mas, antes, precisamos aprender algo essencial em programação: **comentar um código**.

- **Comentar um Código – Caractere #**

Uma das coisas mais essenciais que você precisa aprender em programação, é comentar seu código.

"Lá vai o Python, começa a executar. Python toca pra variável, a variável chama a input, a input passa pra função int que cruza pra função print e exibe o resultado na tela, é gol!"

Não, nada a ver, você não vai comentar nem narrar nada. Comentar é escrever coisas no seu código, em sua língua (português, inglês...) de modo que quem vai ver o código, consiga ler algo.

Vamos voltar no tempo e digitar o Hello, World.

Escreva e rode:

```
# O programa abaixo exibe uma mensagem na tela
print("Hello, World!")
```

Note que tem algo diferente. Na primeira linha.

Escrevemos o símbolo de jogo da velha e escrevemos algo.

Porém, quando vamos rodar nosso script, essa linha não aparece.

É isso que é um comentário: uma linha que começa com o caractere #
O Python simplesmente ignora tudo que começa com #

Pode colocar qualquer coisa, qualquer comando, até uma letra da Xuxa depois de # que o Python não vai ler, não vai rodar aquela linha nem vai aparecer nada no interpretador.

Agora pode parecer bobagem usar isso, mas a medida que seus programas forem ficando enormes, complexos e difíceis, é **altamente recomendável** que você vá comentando seu código, tipo

```
# Aqui estamos conectando com o site da NASA  
[código]
```

```
# Agora vamos tentar invadir o sistema da NASA  
[código]
```

```
# Tendo acesso ao banco de dados da NASA  
[código]
```

```
# O código a seguir começa a baixar arquivos secretos da NASA  
[código]
```

```
# Caceta! Fudeu! Me pegaram
```

Isso ajuda a você quando ver o código de novo, a entender o que cada parte do código está fazendo, para que serve. Ok ?

Sempre comente seus códigos!

Vamos dar um exemplo agora de código comentado, fazendo a calculadora em Python.

- **Calculadora Simples em Python**

O código de nossa calculadora, todo bonitinho, organizado e comentado ficou assim:

```

# Inicialmente, pedimos os dois números para o usuário
# Vamos transformar ele em float
var1 = float( input("Digite um numero: ") )
var2 = float( input("Digite outro numero: ") )

# Calculando a soma e armazenando na variável 'soma'
soma = var1 + var2

# Calculando a subtração e armazenando na variável 'subtracao'
subtracao = var1 - var2

# Calculando a multiplicação e armazenando na variável 'mult'
mult = var1 * var2

# Calculando a divisão e armazenando na variável 'div'
div = var1 / var2

# Calculando a exponenciação e armazenando na variável 'expo'
expo = var1 ** var2

# Calculando o resto da divisão e armazenando na variável 'resto'
resto = var1 % var2

# Imprimindo tudo
print('Soma:      ', var1,'+',var2,' = ', soma)
print('Subtração:   ', var1,'-',var2,' = ', subtracao)
print('Multiplicação: ', var1,'*',var2,' = ', mult)
print('Divisão:      ', var1,'/',var2,' = ', div)
print('Exponenciação:  ', var1,'**',var2,' = ', expo)
print('Resto da divisão: ', var1,'% ',var2,' = ', resto)

```

E o seu ?

- Código Menor da Calculadora em Python

Uma das coisas mais comuns de ocorrer com quem está começando a programar, é criar códigos longos, bagunçados e confusos.

As vezes você programa 200 linhas de código para fazer algo. Aí vem um filho de uma mãe e faz um código melhor, sem erros, mais organizado, que roda mais rápido e usando apenas 30 linhas.

Dá uma raiva...mas, calma. É assim mesmo no começo. O código a seguir é o mesmo da calculadora, porém ele é menor, roda mais

rápido e consome menos memória, pois usamos apenas duas variáveis 'var1' e a 'var2':

```
# Recebe dados do usuário
var1 = float( input("Digite um numero: ") )
var2 = float( input("Digite outro numero: ") )

# Imprime o resultado das operações direto na função print
print('Soma: ', var1, '+', var2, ' = ', var1+var2)
print('Subtração: ', var1, '-', var2, ' = ', var1-var2)
print('Multiplicação: ', var1, '*', var2, ' = ', var1*var2)
print('Divisão: ', var1, '/', var2, ' = ', var1/var2)
print('Exponenciação: ', var1, '**', var2, ' = ', var1**var2)
print('Resto da divisão: ', var1, '%', var2, ' = ', var1%var2)
```

Bacana, não é verdade?

Precedência de Operadores

Você tem uma tia professora línguas.

Ela é tão fera, que é professora tanto de Português como de Inglês.

Mas ela tem uma turma muito grande, e dá muito trabalho ficar somando e dividindo por 2, pois geralmente fica número quebrado no meio.

Vamos ajudar sua tia?

Crie um script que pede a nota de Português e a de Inglês, e forneça a média das duas notas.



Sua tia. Sim, sua tia toca baixo, qual o problema ?

- **Calculando Média em Python**

Vamos criar a variável 'port', para a notar de Português e armazenar um número float nela, através das funções float() e input().

Depois, fazemos o mesmo para a variável 'ing', de Inglês.

Por fim, somamos as duas e dividimos por 2. O resultado dessa operação, armazenamos na variável 'media' e simplesmente damos um print mostrando a média.

Digite e teste o código abaixo:

```
# Pedindo as notas
port = float( input("Digite a nota de Português, tia: ") )
ing = float( input("Tia, agora a de Inglês: ") )

# Cálculo da média
media = port + ing / 2

print("A média desse estudante foi: ",media)
```

Vamos ver o resultado dando uma nota 8 e uma 10 (logo a média deve dar 9):

```
Digite a nota de Português, tia: 8
Tia, agora a de Inglês: 10
A média desse estudante foi: 13.0
>>> |
```

Peraí!!!! Média 13 ???



O que foi isso que aconteceu? Que tiro foi esse, viado ???
 Calma, por incrível que pareça, este resultado está correto.

Sim, correto. Mas do ponto de vista do Python. Do seu, que quer calcular uma média, tá errado...e o grande culpado dessa confusão é um sujeito chamado "Precedência de operadores".



Sua tia ficou chateada com o bug no
 seu programa

- **Precedência de operadores em Python**

Quanto dá: $1 + 1 * 2$?

Alguém pode pensar: $1 + 1 * 2 = 1 + 2 = 3$

Outro pode fazer: $1 + 1 * 2 = 2 * 2 = 4$

E aí, qual o certo?

Pra não haver essa confusão, o Python tem uma regra, sobre qual operação fazer primeiro.

Na ordem de importância, maior pra menor:

- Exponenciação: `**`
- Multiplicação, divisão e resto da divisão: `*` / `%`
- Adição e subtração: `+` -

Tem exponenciação? Não, ok.

Tem multiplicação, divisão ou resto da divisão? Sim, tem multiplicação.

Então multiplica primeiro $1 * 2 = 2$

Só depois faz a soma: $1 + 2 = 3$

Parêntesis pra evitar confusão

Vamos rever nosso código da média:

- `media = port + ing / 2`

Huum! Taí o erro! Primeiro ocorre a divisão da nota de inglês por 2.

Mas, na verdade, a gente queria que a soma ocorresse antes, só depois divide tudo por 2.

Uma maneira de resolver isso é simplesmente colocando parêntesis, pra deixar bem claro o que deve ser feito.

Assim, o correto seria:

```
# Pedindo as notas
port = float( input("Digite a nota de Português, tia: ") )
ing = float( input("Tia, agora a de Inglês: ") )
```

```
# Cálculo da média  
media = (port + ing) / 2  
print("A média desse estudante foi: ",media)
```

Aaaaaagora sim! Vai dar direitinho!



Agora sua tia ficou feliz!

◦ Exercício: Média e Precedência

Devido a crise que está ocorrendo no Brasil, demitiram o professor de Matemática.

E adivinhe para quem sobrou? Claro, pra sua tia. Agora ela vai dar aula de Matemática também.

Faça um programa em Python que receba as notas de Português, Inglês e Matemática de um aluno, e em seguida forneça a média aritmética dessas notas.

Não se esqueça de usar parêntesis no cálculo da média, para não ter problemas de precedência de operadores.

Formatando números na função print

Você foi contratado pela Caixa Econômica Federal para atualizar seu sistema.

Como primeira tarefa

a, você vai criar um script que diga quanto cada ganhador da Mega-Sena vai receber, ao ter o prêmio dividido com outros ganhadores.

Solução:

Faça com que o programa peça um valor (float) ao usuário, que é um valor em dinheiro do prêmio.

Em seguida, peça o número de pessoas (inteiro) que acertaram.
Por fim, exibimos quanto cada um vai ganhar.

Nosso código fica:

```
# Prêmio da Mega-Sena
total = float( input('Premio total da Mega: ') )

# Número de ganhadores
num = int( input('Numero de ganhadores: ') )

print('Cada um vai ficar com R$ ', (total/num) )
```

Mas veja como ficou a saída quando colocamos o prêmio de 1 milhão de reais para 3 pessoas:

```
Premio total da Mega: 1000000
Numero de ganhadores: 3
Cada um vai ficar com R$  333333.3333333333
>>> |
```

Mas...que número feio! R\$ 333333.3333333333 ?

- Formatando Números em Python

Ninguém escreve com tantas casas decimais, nem existe esse valor pequeno de centavos.

O correto é ter apenas duas casas decimais, ou seja, dois números depois da vírgula.

Para isso, vamos usar o operador **%f**, que representa um decimal que iremos formatar (mudar a forma como é exibido).

Por exemplo, se eu usar **%.2f**, o Python vai entender que quer duas casas decimais após o ponto.

Se usar **%.1f**, o Python exibe só uma casa decimal.

Colocamos esse treco aí dentro da função print.
Mas, calma, não vai aparecer isso aí, e sim o número que colocamos na função print.

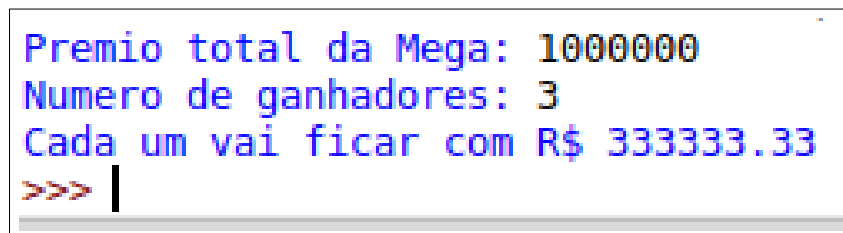
Veja um exemplo correto do código anterior:

```
# Prêmio da Mega-Sena
total = float( input('Premio total da Mega: ') )

# Número de ganhadores
num = int( input('Numero de ganhadores: ') )

print('Cada um vai ficar com R$ %.2f' % (total/num) )
```

Pronto, agora o resultado vai ser:



```
Premio total da Mega: 1000000
Numero de ganhadores: 3
Cada um vai ficar com R$ 333333.33
>>> |
```

Veja agora este outro exemplo de formatação:

```
# Prêmio da Mega-Sena
total = float( input('Premio total da Mega: ') )

# Número de ganhadores
num = int( input('Numero de ganhadores: ') )

print('O premio total foi R$%.2f para %d ganhadores, onde cada um ficou \
com R$%.2f' % (total,num,total/num))
```

Usamos **%.2f** duas vezes, para os valores em dinheiro, decimais.
Para o número inteiro 'num', se usa **%d**

Exercícios Básicos de Python

Se chegou até aqui, nossos sinceros parabéns. A equipe do **Curso Python Progressivo** fica muito feliz! Mas quando ficar milionário, não esquece da gente, ok?

Foram vários tutoriais, bem longos e com bastante informação. Esta altura do campeonato, você já está bem encaminhado em programação.

Para consolidar seus conhecimentos, iremos propor uma série de exercícios bem interessantes e legais de se fazer, com problema reais, do cotidiano, que você irá encarar no seu dia-a-dia como programador Python.

◦ Exercícios Básicos de Python

01. Escreva um programa que pede o raio de um círculo, e em seguida exiba o perímetro e área do círculo.

Para saber o valor do pi, faça:

```
import math  
print(math.pi)
```

Pronto, para saber o valor de pi, basta usar 'math.pi', que é um float

02. Você está no Brasil, e para temperatura usamos o grau Celsius.

Porém, quando você for contrato para trabalhar como programador Python no exterior, deverá usar graus Fahrenheit.

A fórmula da conversão é a seguinte:

$$F = \frac{9}{5} C + 32$$

Ou seja, você fornece a temperatura em graus Celsius, e seu script faz a conversão para graus Fahrenheit.

03. Agora faça o contrário. Você fornece a temperatura em graus Fahrenheit, seu programa conversa para Celsius e exibe na tela.

04. Um novo modelo de carro, super econômico foi lançado.
Ele faz 20 km com 1 litro de combustível.

Cada litro de combustível custa R\$ 5,00.

Faça um programa que pergunte ao usuário quanto de dinheiro ele tem e em seguida diga quantos litros de combustível ele pode comprar e quantos quilômetros o carro consegue andar com este tanto de combustível.

Seu script será usado no computador de bordo do carro.

Testes Condicionais

Parabéns por ter concluído o [módulo básico do curso de Python](#) !

Agora você já sabe criar script, sabe o que são strings, inteiros, floats, sabe fazer operações matemáticas, receber dados do usuário e já fez vários exercícios e programas legais.

Neste módulo de nosso curso, vamos deixar nossos scripts mais complexos, fazendo com que eles tomem decisões, executem de maneira diferente de acordo as ações do usuários.

Também aprenderemos a criar *loopings*, na próxima seção, fazendo nossos programas ficarem em laços repetitivos, fazendo coisas bem interessantes.

Operadores de Comparação em Python: ==, !=, >, <, >=, <=

Já falamos em um tutorial passado, que a função básica do computador é computar, ou seja, fazer contas, cálculos.

Outro pilar da computação é a comparação.

Você pode não acreditar agora, mas a tecnologia hoje é o que é por conta da capacidade dos computadores de contar e fazer comparações.

No término deste módulo, sobre teste condicionais e laços, você vai entender bem isso.

Mas antes de estudar estes assuntos, precisamos conhecer alguns operadores que fazem comparações.

- **Operador de igualdade: ==**

Quando queremos comparar dois valores, usamos o operador de igualdade, que é: ==

Por exemplo, ao escrever 'x == y' estamos fazendo uma comparação, uma pergunta: x é igual a y?

Vamos fazer algumas comparações no interpretador do Python, e ver o resultado que ele nos dá:

```
>>> 1 == 1
True
>>> 2 == 2
True
>>> 1 == 2
False
>>> |
```

Basicamente, o que fizemos perguntar ao Python: '1 é igual a 1?', '2 é igual a 2?' e '1 é igual a 2'? E ele foi nos respondendo com True ou False.

Cuidado para não confundir:

x=1 : aqui estamos fazendo uma atribuição, dizendo que a variável x vai ter o valor 1 e ponto final. Acabou. É uma afirmação.

`x==1` : aqui é uma comparação, estamos perguntando ao Python: 'Pyzinho querido, x tem valor 1?' E ele responde com verdadeiro ou falso. Ok ?

- **Operador de diferente: `!=`**

O contrário do operador 'igual a' é o operador 'diferente de', representado por `!=`

Este operador compara duas coisas, e retorna True se forem diferentes e False se forem iguais.

```
>>> 1 != 2
True
>>> 1 != 1
False
>>> |
```

- **Operador maior que: `>`**

Esse e os próximos operadores, você vai se lembrar da escola.

`>` é o operador 'maior que'.

Quando fazemos: `x > y`, estamos querendo saber se x é maior que y.

Se for verdade, retorna True.

Se x por igual ou menor que y, retorna False.

- **Operador menor que: `<`**

Ao fazer: `x < y`, estamos perguntando ao Python se x é menor que y.

Se ele for menor mesmo, retorna True.

Se for igual ou maior que y, retorna False.

```
>>> 3 > 1
True
>>> 3 > 3
False
>>> 1 < 2
True
>>> 2 < 2
False
>>> |
```

- **Operador maior igual a: \geq**

Seja a operação: $x \geq y$

Você deve ler: x maior ou igual a y ?

Esse operador vai retornar True (verdadeiro) se x for igual a y ou se x for maior que y.

Vai retornar False apenas se x for menor que y.

Operador menor igual a: \leq

$x \leq y$

Você deve ler: x menor ou igual a y ?

Retorna True se x for igual a y ou menor.

Se x for maior que y, retorna False.

```
>>> 2 >= 2
True
>>> 2 >= 3
False
>>> 1 <= 2
True
>>> 1 <= 1
True
>>> 1 <= 0
False
>>> |
```

- **Operadores de Comparação: Tabela**

$x == y$: x é igual a y ?

$x != y$: x é diferente de y ?

$x > y$: x é maior que y?

$x < y$: x é menor que y ?

$x \geq y$: x é maior ou igual a y ?

$x \leq y$: x é menor ou igual a y ?Prontinho. Bem simples, fáceis e óbvios de se entenderem, os operadores de comparação, não é verdade?

Parece até algo 'bobo' demais, mas você vai ver que vai ser capaz de fazer muita, mas muita maldade mesmo, usando estes operadores.

IF - Teste Condicional em Python

Este é, sem dúvidas, um dos tutoriais mais importantes de nosso **Curso de Python**, onde iremos falar sobre como usar o teste condicional IF .

- **Estruturas de Decisão**

Quando uma pessoa liga o computador para usar, o que acontece?

A resposta é: **depende**.

Algumas vão clicar no navegador e acessar o Facebook.

Outra pessoa vai abrir um jogo.

Muitos vão abrir o Office para trabalhar.

Outros vão abrir um programa que reproduz mp3, e por aí vai.

Ou seja: o que o computador vai fazer depende do que o usuário quer fazer.

Em nossos programas, em Python, até agora, todos os scripts foram executados da mesma maneira. Do começo pro fim, de cima pra baixo, sempre do mesmo jeito.

Agora, usando o teste condicional **if**, vamos aprender como um programador pode tomar rumos diferentes de acordo com as decisões do usuários.

- **if: Teste Condicional em Python**

A nomenclatura do teste condicional if em Python é:

if condition:

[código]

[código]

 ...

if em inglês significa 'se'.

Onde tem escrito 'condition', colocamos uma condição, uma comparação, uma espécie de pergunta, que caso seja verdadeira, executa o código abaixo do if.

Se for falsa a condição, não executa o código abaixo do if.

Simples assim.

- **Exercício com teste condicional IF em Python**

Escreva um programa que pede a idade do usuário.

Se ele for maior de idade, exibimos uma mensagem dizendo que já pode dirigir.

O código de nosso programa é:

```
idade=int(input('Idade: '))
```

```
if idade>17 :  
    print('Você é de maior, já pode dirigir!')
```

Veja que nosso teste condicional foi:

`idade > 17`

Ou seja, se a idade que você digitou for 18 ou mais, o código dentro do **if** seria executado. No caso, o código é um **print** na tela, de uma mensagem.

No lugar de 'idade>17' poderíamos ter usado 'idade>=18' que teria o mesmo efeito: só quem tem 18 anos ou mais vai visualizar a mensagem.

- **Indentação em Python**

Para que o nosso código com o teste condicional **if** funcione, é necessário haver a indentação, que nada mais é que dar o espaçamento dentro do **if**.

O correto é:

```
if age>17:  
    print('Você é de maior, já pode dirigir!')
```

Se fizermos:

```
if age>17:  
print('Você é de maior, já pode dirigir!')
```

Não vai funcionar, pois o código dentro do **if** não está indentado (espaçado na linha abaixo do if).

Essa é uma das características do Python.

Ele te obriga a indentar. Sempre que for fazer um teste condicional **if**, no código dentro dele, você tem que dar pelo menos um *espaço*, ok?

- **Como usar o **if** em Python**

Faça um programa que pergunta o gênero da pessoa. Se ela for mulher, digite 1. Se for homem, digite 2. Outro, 3.

Para cada um, ele deve exibir uma mensagem dizendo o gênero escolhido.

Para resolver este exercício, vamos usar três testes condicionais **if**.

Um para testar se é mulher, outro para testar se é homem e outro para testar se é outro gênero.

Nosso código fica:

```
sexo = int( input('Digite 1 se for mulher, 2 homem ou 3 outro: ') )
```

```
if sexo == 1:  
    print('Você é mulher')  
if sexo == 2:  
    print('Você é homem')  
if sexo == 3:  
    print('Outro gênero')
```

Qual o resultado do script acima?

Não tem como saber, vai depender do usuário! O programa roda de uma maneira diferente para cada tipo de situação.

IF e ELSE - Instrução de Teste em Python

No artigo anterior, estudamos o teste condicional IF em Python e vimos o quanto ele é útil e interessante, pois o programa pode tomar diversos rumos diferentes, de acordo com as informações que o usuário vai fornecer.

Agora vamos estudar outra instrução, a **ELSE**, que é usada com a **IF** e deixa nossos testes cada vez mais poderosos.

- **IF e ELSE em Python**

Relembrando o último tutorial de nosso **Curso de Python**, a estrutura de um teste condicional IF tem a seguinte forma:

```
if [condition]:  
    [codigo]    # Se a condição for verdadeira  
    [codigo]    # o código indentado abaixo do if  
    [codigo]    # será executado
```

Ou seja, se a condição *[condition]* for verdadeira, o código abaixo do IF, que está indentado, será executado.

Se a condição for falsa, todo esse bloco do IF é pulado e nada dessas linhas é executado.

Em outras palavras, se a condição for **TRUE**, o código é executado. E é aí que entra a instrução **ELSE**, ela vai ficar sempre abaixo do seu IF, alinhada, e abaixo do **ELSE** (e indentado) teremos um outro código, que só será executado se a condição testada for **FALSE**.

Veja a estrutura de uma teste condicional IF ELSE:

```
if True:  
    [codigo]    # Se a condição for verdadeira  
    [codigo]    # o código indentado abaixo do if  
    [codigo]    # será executado  
else:  
    [codigo]    # Este bloco de código só funciona  
    [codigo]    # se a condição for falsa
```

Vamos ver dois exemplos.

Abra seu IDLE e teste os códigos, veja os resultados você mesmo.

Exemplo de IF e ELSE:

Em programação, o número 1 é sinônimo de verdadeiro e o 0 é sinônimo de falso. Ou seja, 1 é True e 0 é False.

```
if 1:
    print("Oi, estou dentro do if")
else:
    print("Eu nunca vou ser printado :( ")
```

O código acima vai sempre executar o IF, pois o número 1 retorna TRUE. Lembre-se: ali depois IF é um teste, é uma condição, e testes sempre retornam verdadeiro ou falso.

Agora uma condição que retorna sempre falso:

```
if 0:
    print('Somente programadores podem ler isso')
else:
    print('Oi mãe, to no ELSE')
```

Veja que o ELSE é que é executado desta vez.

Exemplo de IF e ELSE:

Escreva um código que exiba o nome de dois times, em seguida pergunta ao usuário qual deles é o melhor.

Com a resposta em mãos, usando IF e ELSE, imprima na tela uma mensagem dizendo qual time ele torce.

Veja nosso código:

```
print('1. Corinthians')
print('2. Flamengo')
resposta = int (input('Qual melhor time: '))

if resposta == 1:
    print('Você deve ser corinthiano')
```

```
else:  
    print('Você deve torcer Flamengo')
```

Perguntamos o time do usuário, e armazenamos na variável 'resposta'.
Em seguida, vamos fazer um teste com essa variável, na instrução **IF**.

Fizemos 'resposta == 1'.

Se o resultado desse teste for verdadeiro, cai no IF e falamos do Corinthians.
Se for falso, cai no ELSE e falamos do Flamengo.

- **Hackeando em Python**

Uma das coisas que iremos te ensinar no decorrer de nosso curso, é hackear.

Ou seja, achar brechas, problemas e falhas em códigos.

Vamos te apresentar duas, agora.

A primeira é simples, quando o nosso programa for pegar o dado de seu teclado, digite qualquer coisa que não seja um número, como uma letra e veja o que ocorre:

```
1. Corinthians  
2. Flamengo  
Qual melhor time: p  
Traceback (most recent call last):  
  File "/home/user/if.py", line 3, in <module>  
    resposta = int(input('Qual melhor time: '))  
ValueError: invalid literal for int() with base 10: 'p'  
>>> |
```

O erro aconteceu na função int(), que esperava receber um valor numérico e recebeu um caractere. Mais pra frente, vamos aprender como contornar isso, garantir que o dado que o usuário forneça seja o que realmente estamos desejando, senão isso pode dar brechas de seguranças em nossos programas e sistemas em Python.

O outro problema que encontramos é usando números mesmo.

Digite, por exemplo, o valor 3 ou qualquer outro que não seja 1 ou 2:

```
1. Corinthians  
2. Flamengo  
Qual melhor time: 3  
Você deve torcer Flamengo  
>>> |
```

Opaaa! Pera lá!

Digitei 3 e apareceu que devo torcer Flamengo?

Por que apareceu isso?

Eu não torço nem dei a entender que queria isso como resposta, por que apareceu esta mensagem?

Você consegue identificar o motivo disso ocorrer?

No próximo tutorial de nosso módulo vamos explicar melhor isso e, claro, consertar essa falha.

Estruturas Aninhadas - IF ELSE dentro de IF ELSE

No tutorial de Python anterior de nosso, falamos em mais detalhes sobre a estrutura de controle IF e apresentamos a sua companheira ELSE.

Fizemos um programinha simples, e hackeamos ele, encontrando alguns problemas.

- **Corrigindo Bugs**

No tutorial anterior, propomos uma questão:

O usuário deve dizer qual o maior e melhor time de futebol do mundo.

Se digitar 1, diz que é o Corinthians.

Se digitar 2, afirma que o Flamengo é tal clube.

Nosso código Python ficou assim:

```
print('1. Corinthians')
print('2. Flamengo')
resposta = int (input('Qual melhor time: ') )

if resposta == 1:
    print('Você deve ser corinthiano')
else:
    print('Você deve torcer Flamengo')
```

Mas aí notamos um problema: se você digitar 3, 4, -1, 0 ou qualquer outro número que não seja 1, diz que você torce Flamengo.

Isso está errado (nada contra o Flamengo), é que se você digitou 3, 4 ou outro número, você não está escolhendo o Flamengo, então por que raios aparece que sou torcedor do Flamengo?

A resposta é simples.

Se você digitar 1, o teste condicional do IF vai dar verdadeiro e cai no print do Corinthians. Até aí ok.

Mas para **qualquer outro número** que digitar, vai cair no ELSE e falar do Flamengo!

Isso é um *bug*, uma falha. Vamos consertar!

Devemos incrementar o seguinte ao nosso programa: qualquer outro número que não seja 1 ou 2, o script deve exibir a mensagem na tela "É, você não deve ser torcedor de Corinthians o Flamengo".

Pronto, agora sim vai ficar bacana.

Vamos resolver esse bug de nosso código.

- **Estruturas Aninhadas - IF ELSE dentro de IF ELSE**

Veja que, inicialmente, fazemos o teste no código, pra verificar se a resposta foi 1. Se sim, ok, fala do Corinthians.

Quando a resposta não for 1, devemos fazer outro teste: checar se o número digitado foi 2. Se for, exibir a mensagem do Flamengo. Se não (**else**), exibir a mensagem que o usuário deve torcer outro time.

E como fazer esse outro teste?

Ué, usando **IF** e **ELSE**.

A diferença é que vamos fazer esse outro teste dentro o **else**,

Ou seja, estamos aninhando as estruturas de controle IF e ELSE, colocando uma dentro da outra.

Nosso código vai ficar assim:

```
print('1. Corinthians')
print('2. Flamengo')
resposta = int(input('Qual melhor time: '))

if resposta == 1:
    print('Você deve ser corinthiano')
else:
    if resposta == 2:
        print('Você deve torcer Flamengo')
    else:
        print('Certamente você não torce Corinthians nem Flamengo')
```

Nossa lógica é a seguinte: Se digitar 1, ok é corintiano.

Se não for 1, vai pro ELSE e vamos fazer outro teste.

Dentro do ELSE, colocamos um IF pra testar se ele digitou 2.

Se sim, ok é flamenguista.

Se não for, cai dentro do ELSE, mas desse novo ELSE, ok?

O usuário só cai dentro desse novo ELSE se não digitar nem 1 e nem o 2.

Faz sentido, né?

- **Aninhar e Indentar em Python**

Note algumas coisas sobre IF e ELSE.

Para cada IF, temos um ELSE.

Podemos criar um IF sem um ELSE, mas não existe ELSE sem IF

Você tem sempre que alinhar o IF de seu ELSE.

Se usar IF ELSE aninhado, ele deve estar abaixo e um pouco mais pra frente que o IF ou ELSE anterior

O molde de IF ELSE's aninhados é o seguinte:

```
if condição1:      # Primeiro IF
    [codigo]

    if condição2:   # Segundo IF
        [codigo]
    else:           # ELSE é do segundo IF
        [codigo]

else:              # ELSE do primeiro IF
    if condição3:
        [codigo]
    else:          # ELSE do terceiro IF
        [codigo]
```

Não tem limite de aninhamento.

O importante é um estar dentro (aninhado) do outro, e alinhado (indentado).

Ok ?

IF e ELSE aparecem aos pares, e o Python só vai saber a qual IF o ELSE pertence, de acordo com o alinhamento, por isso você deve deixar tudo muito bem aninhado e organizado.

Por isso que dizemos que o código Python é 'organizado', pois somos obrigados a deixar tudo bem alinhado, senão não funciona seu código.

Exercícios Resolvidos de IF e ELSE

Agora que você sabe usar o teste condicional IF, a estruturas de controle IF ELSEbem como aninhar IF e ELSE's, vamos resolver alguns exercícios de Python sobre o assunto, para consolidar melhor nossos conhecimentos, fazendo exercícios bem interessantes e realistas.

Exercícios de IF e ELSE em Python

Antes de mais nada, gostaríamos de frisar novamente a importância de você tentar resolver todos os exercícios por conta própria, sem consultar as respostas e soluções.

Mesmo que não consiga, saia feio, cheio de bugs ou muito grande, **tente**. É tentar que vai te transformar em uma excelente programador Python

1. Faça um programa que peça dois números e imprima o maior deles.
2. Faça um script que peça um valor e mostre na tela se o valor é positivo ou negativo.
3. Crie um programa que verifique se uma letra digitada é "F" ou "M". Conforme a letra escrever: F - Feminino, M - Masculino, Sexo Inválido.

Código comentado dos exercícios

Exercício 01:

Pedimos o primeiro número ao usuário, e armazenamos na variável 'num1'. Em seguida, pedimos o segundo e colocamos na variável 'num2'.

Agora vamos aos testes!

Primeiro, testamos se o primeiro é maior que o segundo: **num1 > num2**
Se for verdade, exibimos que num1 é o maior e pronto.

Se for falso, cai ELSE, então num2 é que deve ser maior, correto?

Errado!

Pode ser que sejam iguais!

Então, precisamos fazer outro teste IF dentro desse ELSE (estruturas

aninhadas!) perguntando se são iguais.

Se forem, exibe a mensagem dizendo que são iguais.

Se não forem iguais, aí sim, é porque num2 é maior e caio no else.

Veja como ficou o código:

```
num1=int( input('Digite o primeiro numero: ') )
num2=int( input('Digite o segundo numero: ') )
```

```
if num1 > num2 :
    print('O primeiro, %d, é maior' %num1)
else:
    if num1 == num2 :
        print('Os números são iguais')
    else:
        print('O segundo, %d, é maior' %num2)
```

Exercício 02:

Vamos solicitar o número ao usuário e armazenar na variável 'num'.

Para saber se é positivo, basta fazer o seguinte teste condicional:

num > 0

Se o resultado for TRUE, entra no IF e dizemos que é positivo.

Se o resultado do teste for FALSE, vai pro else, então é negativo...errado!

Pode ser 0, que não é nem positivo nem negativo.

Então, dentro desse ELSE, vamos criar mais um bloco IF ELSE.

Dentro desse novo ELSE fazemos o teste para saber se é 0:

num == 0

Se for, entramos nesse novo IF e dizemos que o número é 0.

Se não for 0, cai no ELSE aninhado, e aí sim, dizemos que é negativo:

```
num=int( input('Digite um numero: ') )
```

```
if num > 0 :
    print('Positivo')
```

```
else:
    if num == 0 :
        print('Nem positivo nem negativo, é 0')
    else:
        print('Negativo')
```

Exercício 03:

Até o momento, só fizemos comparações com números.
Nesse exercício, vamos trabalhar com letras, ou seja, strings.

E strings tem aquela regra: **é sempre dentro de aspas**. OK?

Então, iniciamos pedindo uma letra ao usuário e armazenamos na variável 'resposta', que vai ser uma string.

No primeiro IF, comparamos para saber se digitaram 'M'.
A comparação de strings é assim: **resposta == 'M'**
Ou seja, sempre entre aspas.

Se o IF for true, ok, dizemos que é masculina.
Se não for?

Ai cai no ELSE e vamos ver se foi digitado 'F'.
Se foi, cai no IF aninhado e dizemos ser feminina.

Se não for, cai no ELSE aninhado, então o maldito usuário não digitou nem M nem F.

```
resposta=input('M ou F: ')
```

```
if resposta == 'M':
    print('Masculina')
else:
    if resposta == 'F':
        print('Feminina')
    else:
        print('Você não digitou M ou F')
```

A instrução ELIF em Python

Já estamos quase experts na arte dos testes condicionais, ou controle de fluxo...enfim, nesse treco de fazer teste, IF, ELSE e tal.

Agora, chegou nossa última lição no assunto: a instrução ELIF. Ela não vai nos dar nenhuma informação nova não, pelo contrário, vem pra facilitar nossa vida!

- **Testes, Testes e Mais Testes**

Uma das instruções, em computação, mais executadas, sem dúvidas é o teste condicional. Seja lá a linguagem que usou pra fazer um programa, está entupido de testes.

Desde o momento que liga seu computador ou celular, milhões e milhões de testes condicionais ocorrem.

Clicou em um botão? Teste. Acontece isso.

Clicou em outro botão? Teste. Acontece aquilo.

Digitou certo a senha? Teste. Entra no e-mail.

Digitou errado a senha? Teste. Diz que tá errada.

O navegador que está usando para ler nossa apostila de Python online? Tá cheeeeeeeio de IF e ELSE.

Tá ouvindo música em algum aplicativo do celular?

Tá entupidaaaaaaço de IF e ELSE.

Pode parecer bobinhos os exemplos de Corinthians e Flamengo de nossos exemplos, mas quando for programador profissional, vai ver que vai continuar usando eles, e do mesmo jeito que aprendeu aqui no Python Progressivo.

Pois bem, se convenceu da importância deles?

IF e ELSE já moram no seu coração ?

Ok, vamos melhorar o uso deles, aprender a usar o ELIF.

- **Exercício de Python com (muitos) IF e ELSE**

A seguir, temos a tabela dos melhores times de 2018, da CBF.

Você deve criar um programa que pede 'Digite um número de 1 até 10', e de acordo com o número fornecido pelo usuário, indicar qual o time está naquela posição do ranking.

Pos.	Clube	Fed.
1	Palmeiras	SP
1	Cruzeiro	MG
3	Grêmio	RS
4	Santos	SP
5	Atlético	MG
6	Corinthians	SP
7	Flamengo	RJ
8	Botafogo	RJ
9	Atlético	PR
10	Internacional	RS

Por exemplo, a pessoa digita 1. O resultado é 'Palmeiras'.
Digitou 2, deve printar 'Cruzeiro' etc.

Resolução:

O código é bem simples, basta fazermos uma série de IF e ELSE, nosso código fica:

```
resposta=int(input('Que colocação no ranking deseja saber: '))

if resposta == 1:
    print('Palmeiras! Vai porco!')
else:
    if resposta == 2:
        print('Cruzeiro')
    else:
        if resposta == 3:
            print('Grêmio')
        else:
            if resposta == 4:
                print('Santos')
            else:
                if resposta == 5:
                    print('Atlético-MG! Vai galo!')
                else:
                    if resposta == 6:
                        print('Timão!')
                    else:
```

```
if resposta == 7:
    print('Mengo!')
else:
    if resposta == 8:
        print('Botafogo')
    else:
        if resposta == 9:
            print('Atlético-PR')
        else:
            if resposta == 10:
                print('Internacional')
            else:
                print('Só temos até o décimo!')
```

- **ELIF - O que é ? Como usar ?**

Note uma coisa no código anterior: a medida que vamos indentando, ele vai indo pra direita...e vai indo...indo...vai indo...

Imagina se fossem 20 times?
30 times ?

O site da CBF tem 214 times brasileiros no ranking.
Como ficaria nosso código ? ENORME, gigante, bagunçado, feio, um verdadeiro cabaré.

Assim não dá, assim não pode!

E é aí que vai em a instrução ELIF.

Note que tem um trecho de código que se repete, e muito:

```
else:
    if [condição]
```

Isso se repete vááááárias vezes em nosso código!

O que vamos fazer é resumir ela, para:

```
elif [condição]:
```

Simples assim! Pegamos a instrução ELSE que está seguida de uma IF, e fundimos ela em uma instrução chamada ELIF.

- **ELIF - Por que usar ?**

Não precisamos indentar (dar espaço)! Podemos colocar uma ELIF embaixo de outra ELIF e nosso código não fica tão extenso horizontalmente (e escrevemos menos).

O nosso código anterior ficaria:

```
resposta=int(input('Que colocação no ranking deseja saber: '))
```

```
if resposta == 1:
    print('Palmeiras! Vai porco!')
elif resposta == 2:
    print('Cruzeiro')
elif resposta == 3:
    print('Grêmio')
elif resposta == 4:
    print('Santos')
elif resposta == 5:
    print('Atlético-MG! Vai galo!')
elif resposta == 6:
    print('Timão!')
elif resposta == 7:
    print('Mengo!')
elif resposta == 8:
    print('Botafogo')
elif resposta == 9:
    print('Atlético-PR')
elif resposta == 10:
    print('Internacional')
else:
    print('Só temos até o décimo!')
```

Bem mais simples, bonito, alinhado e menor.

Exercício usando ELIF em Python - Conversão de notas A, B, C, D e F

Agora que já aprendemos a instrução ELIF do Python, vamos usá-la para resolver um problema bem interessante.

Sabe aquelas notas dos filmes americanos, A, B, C ?
Um aluno tipo Stiffler tirando F e sendo reprovado?

Vamos aprender a fazer essa conversão, desse sistema para o nosso.

- **Exercício de ELIF em Python**

Crie um programa em Python que peça a nota do aluno, que deve ser um float entre 0.00 e 10.0

Se a nota for menor que 6.0, deve exibir a nota F.
Se a nota for de 6.0 até 7.0, deve exibir a nota D.
Se a nota for entre 7.0 e 8.0, deve exibir a nota C.
Se a nota for entre 8.0 e 9.0, deve exibir a nota B.

Por fim, se for entre 9.0 e 10.0, deve exibir um belo de um A.

- **Código comentado usando ELIF**

Inicialmente, usamos a função float() no input, para pegar o número decimal.
Devemos lembrar em computação não usamos vírgula, e sim ponto.

No Brasil, usamos 1,99
Em computação, deveríamos escrever 1.99
Use o ponto!

Após pegar a nota do usuário, vamos aos testes.
Se essa nota for menor que 6.0, mandamos um F, sem choro nem vela.
Acaba aí o programa.

Caso não seja, cai dentro de um ELIF, onde vamos fazer mais testes.

Se for menor que 7.0, significa que é menor que 7.0 e maior que 6.0, logo, exibimos um D.

Se não a nota não for menor que 7.0, vamos para mais um ELIF.

Se for menor que 8.0, então vai ser menor que 8.0 e maior que 7.0, logo é uma nota C que iremos exibir.

Se a nota não for menor que 8.0, vai dentro do último ELIF.

Então, é menor que 9.0 e maior que 8.0, por tanto é uma nota B.

Então, só a resta a nota ser maior que 9.0

Nesse caso, cai no ELSE e então exibimos a nota A !

Veja nosso código:

```
resposta=float(input('Qual a nota [0.0 - 10.0]: '))  
if resposta < 6.0:  
    print('Nota F')  
elif resposta < 7.0:  
    print('Nota D')  
elif resposta < 8.0:  
    print('Nota C')  
elif resposta < 9.0:  
    print('Nota B')  
else:  
    print('Nota A')
```


Operadores Lógicos - AND, OR e NOT

Até o momento, em nossos testes condicionais, fizemos apenas um teste em cada instrução **IF**.

Porém, é possível fazermos vários testes, em uma mesma instrução IF, tudo de uma vez. Mas para isso precisamos usar os operadores lógicos: **and**, **or** e o **not**.

São bem simples e fáceis de entender.
Vamos estudar eles agora!

- **Operador AND em Python**
- Se eu disser pra você que: para ser um bom programador Python você deve estudar bastante teoria e fazer exercícios. O que você faria para ser um bom programador Python?

Estudaria muita teoria ou faria muita exercícios? Ou os dois?
O segredo está no "... **e** ..."

Ou seja, você precisa fazer OS DOIS, se fizer apenas estudar teoria ou só resolver exercícios, não vai ser um bom programador. Tem que fazer OS DOIS!

O **e** é o operador **and**, em Python.
Seja o seguinte teste:

teste condição1 **and** condição2

O teste vai retornar TRUE somente se as DUAS condições forem TRUE!

Se uma delas for falsa, já era, o retorno é falso.

Pode colocar 1 milhão de condições, se usar **and**:

teste condição1 **and** condição2 **and** condição3 **and** condição4....**and** condição1000000

Só vai retornar verdadeiro se TODAS as condições forem verdadeiras!
Se uma delas for falsa, fodeu a porra toda...o resultado é falso.

Exemplo de uso do operador **and**

Para votar, você deve ter entre 18 anos e menos de 65 anos.

Escreva um programa que pergunte sua idade e diga se você é obrigado a votar ou não.

Temos dois testes:

1. Testar se tem 18 anos ou mais
2. Testar se tem 65 ou menos

Para votar, cada um dos testes deve ser **TRUE**.

O nosso código fica:

```
resposta=int( input('Qual sua idade: ') )
```

```
if resposta>=18 and resposta <=65:
```

```
    print('Você é obrigado a votar!')
```

```
else:
```

```
    print('Você não é obrigado a votar')
```

Rode o código e faça um teste.

Veja que para o IF ter resultado **verdadeiro**, ambas as condições devem ser verdadeiras: tanto deve ter 18 anos ou mais **E DEVE** ter 65 anos ou menos.

Essa é a característica do operador **and**.

Todos os testes devem ser **true**, para o resultado ser true.

Se um deles for **false**, o resultado é falso.

- **Operador OR em Python**

Para ter acesso a fila de prioridade, você deve ser idoso, gestante ou cadeirante. Escreva um programa que pergunta a situação do usuário (se é

idoso, se é gestante, se é cadeirante ou nenhum destes) e diga se ele pode ter acesso a fila prioridade ou não.

Para o usuário ter acesso a fila de prioridade, ele deve ser idoso **OU** gestante **OU** cadeirante. Qualquer um desses. Basta que uma dessas condições seja verdadeira, para o teste condicional ser verdadeiro.

Para fazermos isso, devemos usar o operador **or** (ou em português), cujo modelo é o seguinte:

```
if condição1 or condição2 or condição3 or ... :  
    [código] # Caso qualquer condição seja verdadeira  
    [código] # Esse código vai ser executado  
else:  
    [código] # Caso todas as condições sejam falsas  
    [código] # esse código vai ser executado
```

A solução do programa do idoso, gestante ou cadeirante, é:

```
print('1. Idoso')  
print('2. Gestante')  
print('3. Cadeirante')  
print('4. Nenhum destes')  
resposta=int( input('Você é: ') )  
  
if (resposta==1) or (resposta==2) or (resposta==3) :  
    print('Você tem direito a fila prioritária')  
else:  
    print('Você não tem direito a nada. Vá pra fila e fique quieto')
```

Note que fizemos três testes:

- Se é idoso (resposta==1)

- Se é gestante (resposta==2)
- Se é cadeirante (resposta==3)

Se qualquer um deles for verdade, esse **if** vai ser verdade, pois usamos o operador lógico **or** (OU). Pra ser prioridade, OU você tem que ser idoso, OU tem que ser gestante OU tem que ser cadeirante.

Não precisa ser os três, basta um deles ser verdade, que o teste retorna verdade. Faz sentido pra você?

- **Operador NOT em Python**

O operador **not** em Python é o mais simples. Ele pega a expressão e reverte ela.

Se era uma condição TRUE, ela vira FALSE.
Se algo era FALSE, ela vira TRUE.
Basta colocar **not** antes.

Por exemplo, vamos criar um script que pergunta qual a melhor banda do universo, para o usuário.

Se ele **não** digitar 'rush', chama o usuário de herege.
Se digitar, ok, parabeniza. Veja:

```
banda = input('Qual melhor banda do mundo? ')
```

```
if not banda=='rush':
```

```
    print('Herege!')
```

```
else:
```

```
    print('Correto, é o Rush!')
```

Tabela Verdade: Operador AND

Condição A	Condição B	AND (A.B)
True	True	True
True	False	False
False	True	False

False False False

Tabela Verdade: Operador OR

Condição A	Condição B	OR (A + B)
True	True	True
True	False	True
False	True	True
False	False	False

Tabela Verdade: Operador NOT

Condição A	NOT (~A)
True	False
False	True

Exercícios de IF, ELIF e ELSE

1. Faça um Programa que verifique se uma letra digitada é vogal ou consoante.
2. Faça um programa que pede duas notas de um aluno. Em seguida ele deve calcular a média do aluno e dar o seguinte resultado:

A mensagem "Aprovado", se a média alcançada for maior ou igual a sete;
A mensagem "Reprovado", se a média for menor do que sete;
A mensagem "Aprovado com Distinção", se a média for igual a dez.
3. Faça um Programa que leia três números inteiros e mostre o maior deles.
4. Faça um Programa que leia três números inteiros, em seguida mostre o maior e o menor deles.
5. Faça um programa que pede dois inteiro e armazene em duas variáveis. Em seguida, troque o valor das variáveis e exiba na tela
6. Faça um Programa que leia três números e mostre-os em ordem decrescente.
7. Faça um Programa que pergunte em que turno você estuda. Peça para digitar M-matutino ou V-Vespertino ou N- Noturno. Imprima a mensagem "Bom Dia!", "Boa Tarde!" ou "Boa Noite!" ou "Valor Inválido!", conforme o caso.
8. As Organizações Tabajara resolveram dar um aumento de salário aos seus colaboradores e lhe contrataram para desenvolver o programa que calculará os reajustes.

Faça um programa que recebe o salário de um colaborador e o reajuste segundo o seguinte critério, baseado no salário atual:

salários até R\$ 280,00 (incluindo) : aumento de 20%
salários entre R\$ 280,00 e R\$ 700,00 : aumento de 15%
salários entre R\$ 700,00 e R\$ 1500,00 : aumento de 10%
salários de R\$ 1500,00 em diante : aumento de 5% Após o aumento ser realizado, informe na tela:

- o salário antes do reajuste;
- o percentual de aumento aplicado;
- o valor do aumento;
- o novo salário, após o aumento.

9. Faça um programa para o cálculo de uma folha de pagamento, sabendo que os descontos são do Imposto de Renda, que depende do salário bruto (conforme tabela abaixo) e 3% para o Sindicato e que o FGTS corresponde a 11% do Salário Bruto, mas não é descontado (é a empresa que deposita). O Salário Líquido corresponde ao Salário Bruto menos os descontos. O programa deverá pedir ao usuário o valor da sua hora e a quantidade de horas trabalhadas no mês.

Desconto do IR:

Salário Bruto até 900 (inclusive) - isento

Salário Bruto até 1500 (inclusive) - desconto de 5%

Salário Bruto até 2500 (inclusive) - desconto de 10%

Salário Bruto acima de 2500 - desconto de 20% Imprima na tela as informações, dispostas conforme o exemplo abaixo. No exemplo o valor da hora é 5 e a quantidade de hora é 220.

Salário Bruto: (5 * 220)	: R\$ 1100,00
(-) IR (5%)	: R\$ 55,00
(-) INSS (10%)	: R\$ 110,00
FGTS (11%)	: R\$ 121,00
Total de descontos	: R\$ 165,00
Salário Líquido	: R\$ 935,00

10. Faça um Programa que leia um número e exiba o dia correspondente da semana. (1-Domingo, 2- Segunda, etc.), se digitar outro valor deve aparecer valor inválido.

11. Faça um programa que lê as duas notas parciais obtidas por um aluno numa disciplina ao longo de um semestre, e calcule a sua média. A atribuição de conceitos obedece à tabela abaixo:

Média de Aproveitamento Conceito

Entre 9.0 e 10.0	A
Entre 7.5 e 9.0	B
Entre 6.0 e 7.5	C
Entre 4.0 e 6.0	D
Entre 4.0 e zero	E

O algoritmo deve mostrar na tela as notas, a média, o conceito correspondente e a mensagem “APROVADO” se o conceito for A, B ou C ou “REPROVADO” se o conceito for D ou E.

12. Faça um Programa que peça os 3 lados de um triângulo. O programa deverá informar se os valores podem ser um triângulo. Indique, caso os lados formem um triângulo, se o mesmo é: equilátero, isósceles ou escaleno.

Dicas:

Três lados formam um triângulo quando a soma de quaisquer dois lados for maior que o terceiro;

Triângulo Equilátero: três lados iguais;

Triângulo Isósceles: quaisquer dois lados iguais;

Triângulo Escaleno: três lados diferentes;

13. Faça um programa que calcule as raízes de uma equação do segundo grau, na forma $ax^2 + bx + c$. O programa deverá pedir os valores de a, b e c e fazer as consistências, informando ao usuário nas seguintes situações:

Se o usuário informar o valor de A igual a zero, a equação não é do segundo grau e o programa não deve fazer pedir os demais valores, sendo encerrado;

Se o delta calculado for negativo, a equação não possui raízes reais. Informe ao usuário e encerre o programa;

Se o delta calculado for igual a zero a equação possui apenas uma raiz real; informe-a ao usuário;

Se o delta for positivo, a equação possui duas raiz reais; informe-as ao usuário;

PS: digite 'import math' no início de seu script. Para achar a raiz quadrada da variável x, faça: `math.sqrt(x)`

14. Faça um Programa que peça um número correspondente a um

determinado ano e em seguida informe se este ano é ou não bissexto.

15. Faça um Programa que peça uma data no formato dd/mm/aaaa e determine se a mesma é uma data válida.

16. Faça um Programa que peça um número inteiro e determine se ele é par ou ímpar. Dica: utilize o operador módulo (resto da divisão): %

17. Faça um Programa que leia um número inteiro menor que 1000 e imprima a quantidade de centenas, dezenas e unidades do mesmo. Observando os termos no plural a colocação do "e", da vírgula entre outros. Exemplo:

326 = 3 centenas, 2 dezenas e 6 unidades

12 = 1 dezena e 2 unidades Testar com: 326, 300, 100, 320, 310, 305, 301, 101, 311, 111, 25, 20, 10, 21, 11, 1, 7 e 1

18. Faça um Programa para um caixa eletrônico. O programa deverá perguntar ao usuário a valor do saque e depois informar quantas notas de cada valor serão fornecidas. As notas disponíveis serão as de 1, 5, 10, 50 e 100 reais. O valor mínimo é de 10 reais e o máximo de 600 reais. O programa não deve se preocupar com a quantidade de notas existentes na máquina.

Exemplo 1: Para sacar a quantia de 256 reais, o programa fornece duas notas de 100, uma nota de 50, uma nota de 5 e uma nota de 1;

Exemplo 2: Para sacar a quantia de 399 reais, o programa fornece três notas de 100, uma nota de 50, quatro notas de 10, uma nota de 5 e quatro notas de 1.

19. Faça um Programa que peça um número e informe se o número é inteiro ou decimal. Dica: utilize uma função de arredondamento.

20. Faça um Programa que leia 2 números e em seguida pergunte ao usuário qual operação ele deseja realizar. O resultado da operação deve ser acompanhado de uma frase que diga se o número é:

par ou ímpar;

positivo ou negativo;

inteiro ou decimal.

21. Faça um programa que faça 5 perguntas para uma pessoa sobre um crime. As perguntas são:

"Telefonou para a vítima?"

"Esteve no local do crime?"

"Mora perto da vítima?"

"Devia para a vítima?"

"Já trabalhou com a vítima?" O programa deve no final emitir uma classificação sobre a participação da pessoa no crime. Se a pessoa responder positivamente a 2 questões ela deve ser classificada como "Suspeita", entre 3 e 4 como "Cúmplice" e 5 como "Assassino". Caso contrário, ele será classificado como "Inocente".

22. Um posto está vendendo combustíveis com a seguinte tabela de descontos:

Álcool: até 20 litros, desconto de 3% por litro

acima de 20 litros, desconto de 5% por litro

Gasolina:

até 20 litros, desconto de 4% por litro

acima de 20 litros, desconto de 6% por litro Escreva um algoritmo que leia o número de litros vendidos, o tipo de combustível (codificado da seguinte forma: A-álcool, G-gasolina), calcule e imprima o valor a ser pago pelo cliente sabendo-se que o preço do litro da gasolina é R\$ 2,50 o preço do litro do álcool é R\$ 1,90.

23. Uma fruteira está vendendo frutas com a seguinte tabela de preços:

Até 5 Kg

Acima de 5 Kg

Morango R\$ 2,50 por Kg

R\$ 2,20 por Kg

Maçã R\$ 1,80 por Kg

R\$ 1,50 por Kg

Se o cliente comprar mais de 8 Kg em frutas ou o valor total da compra ultrapassar R\$ 25,00, receberá ainda um desconto de 10% sobre este total. Escreva um algoritmo para ler a quantidade (em Kg) de morangos e a quantidade (em Kg) de maçãs adquiridas e escreva o valor a ser pago pelo cliente.

24. O Hipermercado Tabajara está com uma promoção de carnes que é imperdível. Confira:

Até 5 Kg	Acima de 5 Kg
File Duplo R\$ 4,90 por Kg	R\$ 5,80 por Kg
Alcatra R\$ 5,90 por Kg	R\$ 6,80 por Kg
Picanha R\$ 6,90 por Kg	R\$ 7,80 por Kg

Para atender a todos os clientes, cada cliente poderá levar apenas um dos tipos de carne da promoção, porém não há limites para a quantidade de carne por cliente. Se compra for feita no cartão Tabajara o cliente receberá ainda um desconto de 5% sobre o total a compra. Escreva um programa que peça o tipo e a quantidade de carne comprada pelo usuário e gere um cupom fiscal, contendo as informações da compra: tipo e quantidade de carne, preço total, tipo de pagamento, valor do desconto e valor a pagar.

Soluções

1.

Primeiro, pedimos um caractere ao usuário e guardamos na variável 'caractere'.

O normal agora seria compararmos se é uma vogal 'a', 'e', 'i', 'o' ou 'u' ou não. Nós vamos fazer isso sim, porém tem uma pegadinha aí.

Em programação, uma letra minúscula é diferente da sua maiúscula. Logo, além das vogais que citamos, temos que comparar com 'A', 'E', 'I', 'O' e 'U' também.

Nosso código fica:

```
char=input('Digite um caractere: ')

if char=='a' or char=='e' or char=='i' or char=='o' or char=='u' or \
   char=='A' or char=='E' or char=='I' or char=='O' or char=='U':
    print('Vogal')
else:
    print('Consoante')
```

Para um if não ficar muito grande, com muitas condições, quebramos ele em duas linhas. Para isso, basta usar uma barra \ , como fizemos no código.

2.

Pedimos as notas ao usuário e armazenamos nas variáveis 'nota1' e 'nota2'. Não esqueça de transformá-las em decimal, usando a função float()

Em seguida, calculamos a média.

Use parêntesis para não ter problema de precedência de operadores.

Agora vamos aos testes.

É sempre interessante tratar logo a condição mais simples e que exclui logo todo o restante.

Nesse caso, testamos logo se a média é menor que 7.

Se for, diz reprovado e encerra o script.

Caso não seja menor, é porque é maior ou igual a 7.

Aqui temos que fazer outro teste: já sabemos que é 7.0 ou mais, mas esse número é menor 10.0 ?

Se for, então exibe a mensagem de parabéns.

Se não for menor que 10.0, é porque é 10.0 cravado, então exibimos a mensagem de aprovação com distinção.

```
nota1 = float(input('Primeira nota: '))  
nota2 = float(input('Segunda nota: '))
```

```
media = (nota1 + nota2) / 2
```

```
print('Media: ',media)
```

```
if media<7.0:  
    print('Reprovado')  
elif media<10:  
    print('Aprovado')  
else:  
    print('Aprovado com Distinção!')
```

3.

Vamos armazenar os três números que o usuário digitar nas variáveis 'primeiro', 'segundo' e 'terceiro'.

O pulo do gato, nessa questão, é usar uma variável extra, que chamaremos de 'maior'. A função dessa variável é simples: armazenar o maior valor que ela achar.

Inicialmente, fazemos com que 'maior' aponte para a variável 'primeiro', estamos supondo que o número 'primeiro' é o maior:
maior = primeiro

Agora vamos fazer os testes!

Vamos comparar o primeiro número com o segundo.

Se o segundo for maior que o primeiro, a variável 'maior' agora vai receber o valor da variável 'segundo':

```
maior = segundo
```

Se não for maior, então 'maior' ainda está com o valor da variável 'primeira', que definimos no começo. Então a variável 'maior' vai ter sempre o maior valor, entre os dois primeiros números digitados, concorda?

Agora vamos testar se a variável 'terceiro' é maior que o valor 'maior'.

Se for, o novo valor de 'maior' vai ser o terceiro número:
maior = terceiro

Se não for maior que 'terceiro', ela continua com valor anterior e este é o maior de todos.

Nosso código fica:

```
primeiro = int(input('Primeiro numero: '))  
segundo = int(input('Segundo numero : '))  
terceiro = int(input('Terceiro numero: '))
```

```
maior = primeiro
```

```
if (segundo > maior):  
    maior = segundo
```

```
if (terceiro > maior):  
    maior = terceiro
```

```
print('Maior: ',maior)
```

Não se assuste se não entender de cara.

Leia, releia, pense, pense de novo, reflita, chore em posição fetal até entender.

Programar é assim mesmo. O segredo é continuar tentando entender, continuar pensando...

4.

Vamos usar o mesmo código da questão anterior, pra achar o maior número:

Achar o maior número em Python

Em seguida, vamos achar o menor.

A lógica é a mesma de achar o maior, primeiro fazemos com que a variável 'menor' receba o valor do 'primeiro' número.

Em seguida, comparamos se o segundo valor é menor que o primeiro.

Se for, a variável 'menor' agora vai receber o valor de 'segundo'.

Se não for menor, fica como está ('menor' continua com o valor de 'primeiro', que é o menor entre os dois primeiros números).

Agora vamos fazer o mesmo teste com o terceiro número.

Se ele for menor que o valor armazenado em 'menor', fazemos com que 'menor' receba o valor de 'terceiro'.

Nosso código fica:

```
primeiro = int(input('Primeiro numero: '))
segundo = int(input('Segundo numero : '))
terceiro = int(input('Terceiro numero: '))
```

```
# Achando o maior número
maior = primeiro
```

```
if (segundo > maior):
    maior = segundo
if (terceiro > maior):
    maior = terceiro
```

```
print('Maior: ',maior)
```

```
# Achando o menor número
menor = primeiro
```

```
if (segundo < menor):
    menor = segundo
if (terceiro < menor):
    menor = terceiro
```

```
print('Menor: ',menor)
```

5.

Temos duas variáveis: var1 e var2

Vamos fazer com que **var2** receba o valor de **var1**:
var2 = var1

Agora vamos fazer com que **var1** receba o valor de **var2**...epa!
Vai dar erro, pois eu MUDEI o valor de **var2** no comando anterior!
O valor originalmente em **var2** foi perdido quando fiz essa variável mudar de valor.

E agora, José?

Calma, você faz o **Curso Python Progressivo**, e vai aprender a resolver isso.

O segredo é: usar uma variável auxiliar, a **aux**

A função da **aux** é guardar aquele primeiro valor contido em **var2**.
Então, a troca de valores se dá assim:

```
aux = var2  
var2 = var1  
var1 = aux
```

Faz sentido pra você?

Refleta e veja se entender **perfeitamente**, pois esse algoritmo de troca é MUITO importante!

```
var1 = int(input('Primeiro numero: '))  
var2 = int(input('Segundo numero : '))
```

```
print('Variavel 1: ',var1)  
print('Variavel 2: ',var2)  
print('Invertendo...')
```

```
aux = var2  
var2 = var1  
var1 = aux
```

```
print('Variavel 1: ',var1)  
print('Variavel 2: ',var2)
```

6.

O grande segredo desse tipo de algoritmo, é comparar e trocar valor das variáveis, duas a duas.

Por exemplo, vamos pegar a sequência: 10 - 20 - 30

Primeira posição: 10

Segunda posição: 20

Terceira posição: 30

Primeiro vamos comparar a segunda posição com a terceira.

A terceira posição é maior que a segunda? Se for, inverte.

Agora fica: 10 - 30 - 20

Pronto, colocamos em ordem decrescente as duas últimas posições.

Agora vamos comparar a primeira posição com a segunda.

A segunda posição tem um valor maior (30) que a primeira (10)?
Sim! Tem! Então inverte essas duas.
Agora fica: 30 - 10 - 20

Note que ao fazer isso, da direita pra esquerda, pegando duas a duas a posição, jogamos sempre o maior número pro começo da ordem.

Por fim, vamos comparar novamente a segunda com a terceira posição.
A terceira é maior que a segunda? Se for, troca! E é.
Agora fica: 30 - 20 - 10

Algoritmo:

1. Compara as duas últimas posições
2. Compara as duas primeiras posições
3. Compara novamente as duas últimas posições

```
primeiro = int(input('Primeiro numero: '))  
segundo = int(input('Segundo numero : '))  
terceiro = int(input('Terceiro numero: '))
```

```
print(primeiro, '-', segundo, '-', terceiro)
```

```
if(terceiro > segundo):  
    aux = terceiro  
    terceiro = segundo  
    segundo = aux
```

```
if(segundo > primeiro):  
    aux = segundo  
    segundo = primeiro  
    primeiro = aux
```

```
if(terceiro > segundo):  
    aux = terceiro  
    terceiro = segundo  
    segundo = aux
```

```
print(primeiro, '-', segundo, '-', terceiro)
```

Mais a frente, estudaremos o algoritmo de **bubble sort**, onde vamos aprender a ordenar listas de números de qualquer tamanho. A lógica é a mesma: ir comparando dois a dois, do fim pro começo (ou começo pro fim - depende se quer ordem crescente ou decrescente).

8.

Vamos armazenar o salário na variável 'salario' (que original, hein?)

Teremos outras variáveis no decorrer do programa:

- percentual - percentual de aumento aplicado
- aumento - valor em R\$ do aumento aplicado
- salario_novo - novo valor do salário, reajustado pelo aumento

A única coisa que muda, de acordo com o salário fornecido pelo usuário, é o percentual do aumento.

Vamos fazer uma série de testes IF ELIF ELSE pra descobrir qual percentual correto aplicar.

Se for menor ou igual a R\$ 280,00 , percentual será 20%

Se não for, vamos testar para ver se é menor ou igual a R\$ 700,00 - se for, o percentual será de 15%.

Se também não for menor R\$ 700,00, vamos testar para ver se é menor igual a R\$ 1500,00 - se sim, 'percentual' recebe 10 (%).

Se também não for menor que R\$ 1500,00 é porque é maior então aplicamos 5%.

Para saber o valor do aumento em R\$, primeiro pegamos a variável 'percentual' e dividimos por 100.0 (usamos decimal pra mostrar ao Python que essa variável deve ser tratada como um float):

`percentual = percentual / 100.0`

(essa linha quer dizer: o novo valor de 'percentual' é o valor antigo dela dividido por 100).

Agora multiplicamos 'percentual' por 'salario' e temos o aumento do salário em R\$.

Basta somar esse valor ao salário original, que temos o novo salário:

```
salario = float(input('Salário do colaborador: '))
```

```

if (salario <= 280):
    percentual = 20
elif (salario <= 700):
    percentual = 15
elif (salario <= 1500):
    percentual = 10
else:
    percentual = 5

print('Salario original: R$ ', salario)
print('Percentual: ',percentual,'%')

percentual = percentual/100.0
aumento = percentual * salario
novo_salario = salario + aumento

print('Aumento: R$ ',aumento)
print('Novo salário: R$ ', novo_salario)

```

12.

Vamos receber os três lados do triângulo e armazenar nas variáveis **a**, **b** e **c**.

O primeiro teste que fazemos é para saber se a soma de quaisquer dois lados é menor que o terceiro lado. Se for, esses três valores não formam um triângulo e acabou o programa aí, precisa nem testar se é equilátero, isósceles e escaleno.

Se a soma de dois lados quaisquer for maior que o terceiro lado, ok, é triângulo e vamos pro elif.

Agora, vamos testar se é equilátero, para isso comparamos o lado **a** com o **b** e depois o lado **a** com o lado **c**.

Note que não precisamos comparar os lados **b** e **c**, pois se **a** é igual a **b** E TAMBÉM (operador lógico **and**) **a** é igual a **c**, então o lado **b** vai ser automaticamente igual ao lado **c**

Se forem todos iguais, diz que é equilátero e acabou aí.

Se não for equilátero, cai no próximo elif.

Lá vamos testar se ele tem dois lados iguais: $a==b$ ou $a==c$ ou $b==c$ (notem o OU, que é o operador lógico **or**).

Se alguma dessas comparações retornar verdadeiro, o triângulo é isósceles

e acaba aí.

Porém, se não for isósceles, cai no else final.

Pois se não é equilátero nem isósceles, e é um triângulo, tem de ser escaleno.

```
a = float(input('Primeiro lado: '))
b = float(input('Segundo lado: '))
c = float(input('Terceiro lado: '))

# Testando se é triângulo
if (a + b < c) or (a + c < b) or (b + c < a):
    print('Nao é um triangulo')
elif (a == b) and (a == c) :
    print('Equilatero')
elif (a==b) or (a==c) or (b==c):
    print('Isósceles')
else:
    print('Escaleno')
```

13.

Antes de mais nada, vamos relembrar a fórmula de Bháskara para achar as raízes de uma equação do segundo grau, do tipo: $ax^2 + bx + c = 0$:

$$x = \frac{-b \pm \sqrt{b^2 - 4.a.c}}{2.a}$$

O primeiro teste que fazemos é em relação ao coeficiente **a**. Se for 0, não é uma equação do segundo grau e acaba o programa.

Se for diferente de 0, cai no **else**, que é onde todo nosso programa vai funcionar. Primeiro, dentro do else, pedimos o valor dos coeficientes **b** e **c**.

Agora, vamos calcular o delta.

Em Python, fica assim: $\text{delta} = b*b - (4*a*c)$

Agora vamos testar o delta, dentro de um **if** aninhado no **eles** anterior.

Se for menor que 0, encerramos o programa dizendo que as raízes são imaginárias.

Em seguida, usamos um **elif** para testar se delta for 0, se sim valor da raiz será: $\text{raiz} = -b / (2*a)$

Por fim, se não é menor que 0 e o delta não é 0, é porque vai ser sempre maior que 0. Essa condição cai no **eles** aninhado, onde calculamos as raízes

assim:

```
raiz1 = (-b + math.sqrt(delta) ) / (2*a)
```

```
raiz2 = (-b - math.sqrt(delta) ) / (2*a)
```

Nosso código ficou:

```
import math
```

```
print('Equação do 2o grau da forma:  $ax^2 + bx + c$ ')
```

```
a = int( input('Coeficiente a: ') )
```

```
if(a==0):
```

```
    print('Se a=0, não é equação do segundo grau. Tchau')
```

```
else:
```

```
    b = int( input('Coeficiente b: ') )
```

```
    c = int( input('Coeficiente c: ') )
```

```
    delta = b*b - (4*a*c)
```

```
    if delta<0:
```

```
        print('Delta menor que 0. Raízes imaginárias. Tchau')
```

```
    elif delta==0:
```

```
        raiz = -b / (2*a)
```

```
        print('Delta=0 , raiz = ',raiz)
```

```
    else:
```

```
        raiz1 = (-b + math.sqrt(delta) ) / (2*a)
```

```
        raiz2 = (-b - math.sqrt(delta) ) / (2*a)
```

```
        print('Raízes: ',raiz1, ' e ',raiz2)
```

14.

Anos bissextos são aqueles que são múltiplos de 4, como 1996, 2000, 2004 etc (que podem ser divididos por 4 deixando resto 0).

Porém, há uma exceção: múltiplos de 100 que não sejam múltiplos de 400.

Uma das duas condições a seguir deve ser verdadeira:

Condição 1: Ser múltiplo de 4 e não ser múltiplo de 100

Condição 2: Ser múltiplo de 400 (se for múltiplo de 400 automaticamente é de 4)

Logo, temos o código:

```
ano = int(input('Ano: '))
```

```
if (ano%4==0 and ano%100!=0) or (ano%400==0):
```

```
    print('Bissexto')
```

```
else:
```

```
    print('Não é bissexto')
```

15.

Vamos armazenar os dados nas variáveis 'dia', 'mes' e 'ano'.

Para armazenar o valor lógico verdadeiro ou falso, vamos usar a variável booleana 'valido'. Inicialmente fazer ela ser falsa:

valido = False

O grande segredo nesse algoritmo é o mês.

Primeiro vamos testar se o mês digitado tem 31 dias.

São os meses 1, 3, 5, 7, 8, 10 ou mês 12.

Se tiver digitado um desses valores para 'mes', vamos verificar a variável 'dia' é menor ou igual a 31. Se for, a data é válida e fazemos 'valida = True' Se não for, continua sendo False

Agora vamos testar os meses que tem 30 dias.

Eles são os meses 4, 6, 9 e o mês 11.

Nesses meses, temos que avaliar se a variável 'dia' tem um número menor ou igual a 30. Se sim, fazemos 'valida = True'.

Por fim, vamos avaliar o mês mais problemático, o mês 2, fevereiro.

Inicialmente, é preciso verificar se é ano bissexto, se for bissexto a variável 'dia' deve ser testada para saber se o valor digitado é 29 ou menos. Se sim, validamos a data com 'valida = True'

Se não for ano bissexto, testamos a variável 'dia' para saber se o valor digitado foi 28 ou menos. Se for, 'valida = True'

Caso não tenham digitado um número de 1 até 12 em mês, a variável 'valida' continua tendo valor False, pois não caiu em nenhum IF ou ELIF.

Por fim, testamos a variável booleana 'valida'. Se for True, dizemos que a data é válida, se tiver o valor lógico False nela, dizemos que é inválida:

```
dia = int( input('Dia: ') )
mes = int( input('Mês: ') )
ano = int( input('Ano: ') )
```

```
valida = False
```

```
# Meses com 31 dias
```

```
if( mes==1 or mes==3 or mes==5 or mes==7 or \
    mes==8 or mes==10 or mes==12):
    if(dia<=31):
        valida = True
```

```

# Meses com 30 dias
elif( mes==4 or mes==6 or mes==9 or mes==11):
    if(dia<=30):
        valida = True
elif mes==2:
    # Testa se é bissexto
    if (ano%4==0 and ano%400!=0) or (ano%400==0):
        if(dia<=29):
            valida = True
        elif(dia<=28):
            valida = True

if(valida):
    print('Data válida')
else:
    print('Inválida')

```

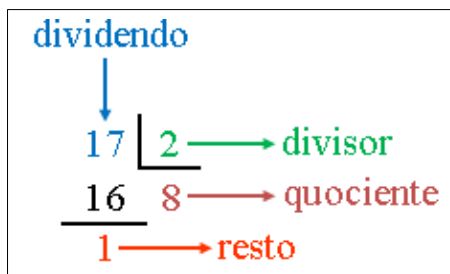
16.

Antes de mostrar o exercício, resolver e comentar o código de como descobrir se um número é par ou ímpar em Python, precisamos revistar um tutorial antigo:

Operações matemáticas em Python

Lá estudamos o operador % que é o operador de resto da divisão. Talvez você não lembre direito que troço é isso de resto da divisão.

Mas vamos voltar pra época da escolinha, quando fazíamos continhas de dividir:



O segredo está ali, no resto. O que sobra. Ele é o segredo de tudo.

Para saber se um número é par ou ímpar, basta dividir ele por 2. Se for par, o resto é sempre 0, não sobra nada. Já se for ímpar, vai sempre ter resto 1.

Saber se é Par ou Ímpar em Python

" Faça um Programa que peça um número inteiro e determine se ele é par ou ímpar. Dica: utilize o operador módulo (resto da divisão): %"

Inicialmente pedimos um número ao usuário, e armazenamos na variável 'numero'. Agora, com um simples teste condicional **IF** vamos verificar o resto da divisão dele por 2.

Se o resultado for 0, é par e cai no print do **IF**, dizendo que é par.

Se não for 0 o resto, é porque vai ser 1 e cai no **ELSE** onde printamos que é ímpar:

```
numero = int(input('Digite um inteiro: '))

if (numero%2) == 0:
    print("Par")
else:
    print("Ímpar")
```

Aperfeiçoando código **Python**

Já dissemos, em algum tutorial anterior, que 1 equivale ao **True** e 0 ao **False**. Quando fazemos o resto da divisão por 2, o resultado é sempre 0 ou 1.

Então nosso código poderia ser assim:

```
numero = int(input('Digite um inteiro: '))

if numero%2 :
    print("Ímpar")
else:
    print("Par")
```

Veja que se o resultado da operação for 1, cai no IF que diz que é ímpar, se for 0 vai pro ELSE que diz que é par.

Bem mais chique, não acha?

Resto da divisão por 3:

"Faça um programa que recebe um inteiro do usuário e diz se esse número é múltiplo de 3 ou não".

Veja o código e tente entender:

```
numero = int(input('Digite um inteiro: '))

if not (numero%3) :
    print("É múltiplo de 3")
else:
    print("Não é múltiplo de 3")
```

Um número é múltiplo de 3 se o resto da divisão dele por 3 for 0.
Então fazemos o teste: `numero % 3`

Se este resultado for 0, o IF não é executado, pois dá falso.
Então o que fazemos? Invertamos com o operador **not**

Sempre que algo for TRUE o not transforma em FALSE, e vice-versa.
Assim colocamos um not antes do `(numero%3)` e quando for múltiplo de 3, a expressão **not (numero%3)** vira TRUE e cai no IF dizendo que é múltiplo de 3.

Fodástico, não ?

Curiosidade sobre resto da divisão %

O resto da divisão de um número por 2 pode ser:
0 ou 1

O resto da divisão de um número por 3 pode ser:
0, 1 ou 2

O resto da divisão de um número por 4 pode ser:
0, 1, 2 ou 3

O resto da divisão de um número por 5 pode ser:
0, 1, 2, 3 ou 4

...

...

...

O resto da divisão de um número por n pode ser:
0, 1, 2, 3, ..., (n-1)

17.

Para sabermos o valor da unidade de um número inteiro positivo qualquer, em Python, basta usar o operador de resto da divisão % da seguinte maneira:

`unidade = numero % 10`

Pronto. Só isso.

Se nosso número é 123 e fizermos `123 % 10`, o resultado vai ser 3. Então 3 é nossa unidade.

E como achar a dezena?

O próximo passo é transformar nosso número de 123 pra 12 (excluimos a unidade).

Primeiro, subtraímos do número o valor da unidade.

$123 - 3 = 120$

Depois, dividimos o número por 10:

$120 / 10 = 12$

Prontinho, pra achar a dezena basta fazer: $12 \% 10 = 2$

Em programação Python, fica assim:

`numero = (numero - unidade) / 10` # Passa de 123 pra $123 - 3 = 120 / 10 = 12$

`dezena = numero % 10` # dezena = $12 \% 10 = 2$

E pra achar o valor da centena?

De novo, transformamos de 12 pra 1 (excluimos 2).

Subtraímos a dezena achada: $12 - 2 = 10$

E novamente dividimos por 10: $10 / 10 = 1$

Em programação Python:

`numero = (numero - dezena) / 10`

`centena = numero`

No caso, paramos por aí. Por só queremos achar até a centena.

Nesse ponto, nosso número vai ser o mesmo valor da centena.

Se quiser para números maiores, só seguir essa lógica.

Código comentado em Python

```
numero = int(input('Digite um numero inteiro positivo: '))

# Extraindo a unidade
unidade = numero % 10

# Eliminando a unidade de nosso número
numero = (numero - unidade)/10

# Extraindo a dezena
dezena = numero % 10

# Eliminando a dezena do número original, fica a centena
numero = (numero - dezena)/10
centena = numero

# Fazendo ser inteiros
dezena = int(dezena)
centena = int(centena)
print(centena,'centena(s)',dezena,'dezena(s) e',unidade,'unidade(s)')
```

18.

Há várias maneiras diferentes de darmos 256 reais, ou qualquer outro valor. Poderíamos, por exemplo dar 256 cédulas de 1 real (hoje em dia só moeda, mas antes tinha cédula de 1 real sim).

Mas seria altamente inconveniente sair de um caixa eletrônico com centenas de cédulas. Ou seja, queremos usar o menor número de cédulas possíveis!

Isso é feito da seguinte maneira: dando cédulas de valor alto.

- Notas de R\$ 100,00

O primeiro passo é tentar empurrar o maior número de notas de 100 reais possível. No caso de 256 reais, só podemos dar 2 notas de 100 reais, pois se dermos mais, passa de 256.

Como fazemos isso?

Dividindo o valor que o usuário pediu pra sacar por 100.

Vamos colocar em 'numero', o valor que ele solicitou que fosse sacado.

Se fizermos:

$cem = numero / 100$

Teremos que $cem = 2.56$

Não seria legal se a gente tirasse esse ponto fluante e essa parte decimal?

É fácil fazer isso, só usar a função **int()**, que vai transformar esse número em inteiro (ele pega a parte inteira e descarta o resto).

Assim, o número de cédulas de 100 reais é:

$cem = \text{int}(numero/100)$

Se antes queríamos sacar 256 reais, agora queremos somente 56, pois já demos os 200 reais.

Vamos transformar nosso valor de 256 pra 56 da seguinte maneira:

$numero = numero - (cem * 100)$

$(numero = 256 - 2 * 100 = 56)$

Concorda?

- Notas de R\$ 50,00

Agora vamos tentar usar o máximo possível de notas de 50 reais.

Para isso, basta dividir nosso número por 50 e pegar a parte inteira, como fizemos com as cédulas de 100:

$cinquenta = \text{int}(numero/50)$

No nosso exemplo de 256 reais: $cinquenta = \text{int}(56/50) = \text{int}(1.12) = 1$

Agora que já usamos as de 50 reais, vamos tirar ela de nosso número:

$numero = numero - (cinquenta * 50)$

$(numero = 56 - 1 * 50)$

- Notas de R\$ 10,00

A lógica é a mesma:

```
dez = int( numero/10 )  
numero = numero - (dez*10)
```

•Notas de R\$ 5,00

```
cinco = int(numero/5)  
numero = numero - (cinco*5)
```

E o que sobrou, é nota de um real:

```
um = numero
```

Nosso código Python fica:

```
numero = int(input('Valor para sacar [10-600]: '))
```

```
cem = int(numero / 100)  
numero = numero - (cem*100)
```

```
cinquenta = int(numero/50)  
numero = numero - (cinquenta*50)
```

```
dez = int(numero/10)  
numero = numero - (dez*10)
```

```
cinco = int(numero/5)  
numero = numero - (cinco*5)
```

```
um = numero
```

```
print('Notas R$100,00 = ',cem)  
print('Notas R$ 50,00 = ',cinquenta)  
print('Notas R$ 10,00 = ',dez)  
print('Notas R$ 5,00 = ',cinco)  
print('Notas R$ 1,00 = ',um)
```

Nosso código também poderia ser:

```
numero = int(input('Valor para sacar [10-600]: '))
```

```
cem = int(numero / 100)  
numero = numero % 100
```

```
cinquenta = int(numero/50)
numero = numero % 50
```

```
dez = int(numero/10)
numero = numero % 10
```

```
cinco = int(numero/5)
numero = numero % 5
```

```
um = numero
```

```
print('Notas R$100,00 = ',cem)
print('Notas R$ 50,00 = ',cinquenta)
print('Notas R$ 10,00 = ',dez)
print('Notas R$ 5,00 = ',cinco)
print('Notas R$ 1,00 = ',um)
```

Consegue entender o motivo?

19. Arredondar números em Python

A utilidade mais básica da função **round()** é receber um número qualquer e arredondar ele. O mais lógico é arredondar um flutuante para um inteiro.

Basta colocar um número dentro dela: `round(numero)` que ela devolve ele arredondado.

Vamos ver alguns exemplos:

```
Numero original: 1.9
Arredondado      : 2

Numero original: 21.12
Arredondado      : 21

Numero original: 3.4
Arredondado      : 3

Numero original: 4.6
Arredondado      : 5
```

O código para você testar a `round()` é:

```
numero = float(input('Numero original: '))
print("Arredondado  :", round(numero))
```

Mas, por exemplo, o número 1.5 ?

Ele está igual distância de 1 como de 2, qual será o resultado se aplicarmos **round(1.5)** ?

A resposta é 2.

Então você deve pensar: *ah ok, arredonda pra cima.*

Então teste **round(2.5)**, o resultado vai ser 3, pra cima, não é?

Errado, é 2 de novo.

"Que bruxaria é essa, Python Progressivo? Tá bugada a função round() ?"
Não!

A função **round()** tem uma característica especial: se o número flutuante estiver igual distância entre o inteiro de cima e o inteiro de baixo, ela arredonda pro **número par** mais próximo!

Veja:

```
Numero original: 1.5  
Arredondado    : 2  
  
Numero original: 2.5  
Arredondado    : 2  
  
Numero original: -3.5  
Arredondado    : -4  
  
Numero original: -5.5  
Arredondado    : -6
```

Arredondar com casas decimais em Python

Colocando apenas um número dentro da round() ela arredondou e nos devolveu um inteiro.

Mas, e se quisermos arredondar pra ter um valor float?

Por exemplo, quero arredondar 1.9999 pra 2.0 ?

Ou 21.129 pra 21.13 ?

A round() faz isso!

Seu estereótipo é: **round(numero, n)**

Onde 'numero' é o número que você seja arredondar e 'n' é o número de casas decimais **após** o ponto flutuante, que deseja arredondar.

Por exemplo:

```
numero = 1.23456789
```

```
round(numero, 1) = 1.2
```

```
round(numero, 2) = 1.23
```

```
round(numero, 3) = 1.235
```

```
round(numero, 4) = 1.2346
```

```
round(numero, 5) = 1.23457
```

```
round(numero, 6) = 1.234568
```

```
round(numero, 7) = 1.2345679
```

```
round(numero, 8) = 1.23456789
```

Precisão cirúrgica, esse Python, não ?

Arredondar pra **baixo** e pra **cima**

Muitas vezes queremos sempre arredondar pra baixo, por exemplo:

```
1.2 -> 1
```

```
1.1 -> 1
```

```
9.9 -> 9
```

Outras vezes podemos querer sempre arredondar pra cima:

```
1.1 -> 2
```

```
1.2 -> 2
```

```
9.9 -> 10
```

Pra arredondar pra baixo um número **num** basta fazer:

```
round(num - 0.5)
```

Para arredondar pra cima:

```
round(num + 0.5)
```

Mas só funciona se for para números decimais, ok ?

Exercício Resolvido de **Python**

Escreva um script que peça um número ao usuário.

Em seguida, ele deve descobrir se o número é inteiro ou decimal.

Se for decimal, deve dizer o número arredondado pra cima e arredondado pra baixo.

Primeiro, vamos descobrir se é inteiro ou decimal. Um número **num** vai ser inteiro quando ele for igual ao seu número arredondado: `num == round(num)` retorna **True** para um inteiro e **False** para um decimal.

Caso seja decimal, usamos `round(num-0.5)` pra arredondar pra baixo e `round(num+0.5)` pra arredondar pra cima.

Nosso código Python fica:

```
num = float(input('Numero original: '))

if num == round(num):
    print("Inteiro")
else:
    print("Decimal")
    print("Arredondado pra baixo: ", round(num-0.5) )
    print("Arredondado pra cima : ", round(num+0.5) )
```


Laços e Loopings: WHILE e FOR

Laços e Loopings - O que são? Para que servem ?

Antes de entrarmos em detalhes sobre um dos assuntos mais importantes de nosso **Curso de Python**, que são os laços (também conhecido por loopings) **while** e **for**, vamos tentar entender a motivação por trás deles.

Em vez de vir e jogar definições e instruções na sua cara, vou fazer entender que alguma coisa deve ser feita/estudada para que nossos conhecimentos possam fazer programas poderosos, grandes e interessantes.

- **Repetindo estruturas e códigos**

"O professor Bruce Dickinson vai dar aula em uma faculdade, é o novo professor e ele precisa que você crie um programa que calcula a média de todos os alunos e diga a média da turma. O professor Bruce tem várias turmas, dos mais variados tamanhos. Resolva esse problema."

"Ah, moleza. Só pedir as notas de cada aluno, somar e calcular pelo tanto de alunos na turma. Até já fizemos exercício disso, Python Progressivo, esqueceu?"

Ok, vamos lá.

Vamos supor que nossa turma tenha 2 alunos.

Nosso código é

media2.py:

```
nota1 = float(input("Nota 1:"))
```

```
nota2 = float(input("Nota 2:"))
```

```
media = (nota1+nota2)/2
```

```
print("A media é: ", media)
```

Já se tiver 3 alunos, será

media3.py

```
nota1 = float(input("Nota 1:"))
nota2 = float(input("Nota 2:"))
nota3 = float(input("Nota 3:"))

media = (nota1+nota2+nota3)/3

print("A media é: ", media)
```

Se a turma tiver 50 alunos, nosso script é...
(imagine aqui um código gigante, que pede 50 notas, vá somando todas e divide tudo por 50).

Prontinho, você manda vários e vários scripts para o professor, para ele usar cada um de acordo com o número de alunos de cada turma.

Ok? Parece fazer sentido, ficou bacana?

- **Programação é simplificação**

NÃO!!! Caramba, que ideia bizarra!

Para cada turma, usar um script diferente? Ia ter o que, uma pasta com 50 programas?

Quando quer usar duas células no Excel, você abre uma planilha.
Para usar três células, você vai abrir outra planilha? Claro que não!
Pode usar uma ou um milhão de células, pode usar a mesma planilha, o Excel não vai te fazer usar milhares de programas diferentes para trabalhar.

O mesmo vale para nosso script.

E se você for contratado para fazer o censo, vai fazer um script com 1 milhão de linhas, pedindo 1 milhão de informação diferente, de cada pessoa...vai usar milhões de variáveis num programa só?

Claro que não! Isso é impossível de se fazer manualmente!

A gente programa pra resolver problemas que seriam difíceis de se fazer na mão. A gente programa pra deixar a vida mais fácil.

E não tenha dúvidas, se estudar pelo **Python Progressivo**, vai fazer muitos e muitos programas úteis, uma nova rede social, um novo Google, um novo Youtube...

- **Progamar é encontrar padrões**

O grande segredo da programação é na sua capacidade de notar padrões. Identificar padrões é o ideal.

Vamos identificar em nosso exemplo?

Primeiro, devemos saber o número de alunos da turma.

E quem sabe isso? O Bruce. Então nosso programa vai pedir a ele esse valor **x** de alunos.

Em seguida, o que fazemos?

Para cada aluno, do primeiro até o de número **x**, vamos pedir um valor e armazenar em uma variável diferente.

Ao final de tudo, nosso script vai somar da primeira variável até a variável que contém o valor da nota do aluno de número **x**.

Depois simplesmente divide tudo por **x**. Essa é a média.

Veja o padrão: o **x** pode ser 1, 2, 3, 4, ..., 2112, ...1 milhão

Tanto faz. A ideia é sempre essa, sempre o mesmo roteiro (*script* em inglês).

- **Laços e Loopings em Python**

Um laço, ou um looping, é algo que se repete.

Vamos estudar os laços **while** e **for**, e chamamos ele de estruturas de repetição.

No nosso exemplo, uma coisa se repete: ficar pedindo as notas.

Vamos fazer com que nossos programas fiquem presos em determinados

loopings, fazendo coisas repetidas (como pedir nota), até onde desejarmos (no caso, deve se repetir x vezes).

Para nós, pedir 1 milhão de dados, usando 1 milhão de linhas de comando, é algo impossível de se fazer na mão.

Mas para os computadores não. Fazem isso em milésimos de segundos.

E o que um programador Python faz?

Manda no computador. Faz ele te obedecer. Cria programas Python que fazem de um computador seu escravo.

Assim, com poucas linhas, você vai programar scripts que fazem milhões de coisas, bilhões...quantas vezes você quiser e mandar.

Você tá ficando foda pra caramba.

Sinta o poder. É você saindo da Matrix e aprendendo a criar sua própria Matrix.

WHILE em Python - Como usar

Neste tutorial de nosso curso de Python, vamos estudar uma das coisas mais importantes e usada em programação: o laço **while** (ou looping while).

- **Laço WHILE**

Sem mais delongas, vamos aprender a usar essa arma de destruição em massa, que é o laço while.

Seu formato é

```
while <teste>:
```

```
    codigo1
```

```
    codigo2
```

```
    codigo3
```

```
...
```

```
    codigo4
```

A instrução **while** funciona assim:

Assim que começa o while, ele faz um teste (como se fosse um IF teste condicional) e testa a instrução **<teste>**.

Se este teste resultar em verdadeiro (TRUE), tudo que está dentro do laço while (codigo1, codigo2, codigo3..., é executado).

Terminou de executar tudo?

Testa de novo. Deu true? Executa tudo de novo...

E fica assim, testando, executando, testando, executando...

Como se fosse um *looping*. E, de fato, é um looping.

E só vai parar quando o teste for FALSE.

Aí acaba o **while** e o 'codigo4', de fora, é executado e o script segue seu percurso naturalmente.

Você vai entender melhor usando alguns exemplos.

- **Exemplo 1** de uso do laço **WHILE**

"Crie um programa que imprima na tela os números 1 até o 10, usando o laço while".

Primeiro de tudo, vamos declarar a variável 'numero' e fazer ela iniciar com valor 1.

O que queremos é imprimir ela, depois fazer ela virar o número 2 e imprimir de novo. Fazer ela receber o número 3, e imprimir de novo...até virar o número 10, imprimir e acabar aí.

O nosso teste é: 'numero é menor ou igual a 10? Ok, então imprime'. E o *looping* é simplesmente imprimir nossa variável.

Nosso código Python fica assim:

```
numero=1  
  
while numero<=10:  
    print(numero)  
    numero += 1
```

Tem uma novidade aí: `numero += 1`
Que é o mesmo que: `numero = numero + 1`

Ou seja, estamos incrementando a variável 'numero' em uma unidade. Vamos ver passo a passo como funciona esse tal de **while**.

Primeiro definimos nossa variável com valor 1.
Depois testamos se ela é menor que 10. Como é, vai executar o laço while.
Primeiro, imprime o valor da variável, que é 1.
Depois adiciona 1 a essa variável.

Agora ela vale 2. O teste é realizado, como 2 é menor que 10.
Executa de novo...imprime 2, incrementa e agora a variável é 3.

E assim vai indo, indo, repetindo, repetindo...vai chegar uma hora que 'numero=10'

Como 'numero<=10' é verdadeiro, o laço while é executado.
O número 10 é impresso na tela, e ele vira 11.

Agora, o teste vai dar falso e o while vai parar de executar.
E prontinho, você imprimiu do número 1 até o 10 usando o laço while!

Simples, não?

Vamos aprofundar mais e deixa esse código mais interessante.

- **Exemplo 2 de uso do laço WHILE**

"Faça um programa que peça um número maior que 1 ao usuário. Em seguida, imprima todos os números, de 1 até o número que o usuário informou".

No exemplo anterior, a gente tinha um 'limite', que era o valor 10.
Quando chegava em 10...créu. Acabava o laço while.

Agora esse valor vai ser informado pelo usuário. Vamos armazenar essa informação na variável 'max' (de máximo).

Nosso código fica:

```
numero=1
max = int(input("Digite um inteiro maior que 1: ") )

while numero <= max:
    print(numero)
    numero += 1
```

Experimente digita 100, 1000, um milhão, um trilhão...

Veja que interessante: quantas vezes esse while vai ser executado?

Ué, depende do usuário!

Pode repetir uma vez, 10, mil, 1 trilhão...há uma interação entre o script e o usuário, vai fazer coisas diferentes de acordo com o desejo do usuário.

- **Exemplo 3** de uso do laço **WHILE**

"Escreva um programa que repita pra sempre, na tela, a mensagem 'Curso Python Progressivo'".

Quando que um laço vai se repetir pra sempre, indefinidamente?

Ué, quando seu teste for SEMPRE verdadeiro!

while True:

```
    print("Curso Python Progressivo")
```

Também poderia colocar '1', '2' ou qualquer outro valor diferente de 0 ou false no teste, que daria sempre verdadeiro e iria sempre repetir.

- **Exemplo 4** de uso do laço **WHILE**

"Crie um programa que peça um número ao usuário e imprima todos os números pares de 1 até o número fornecido"

Vamos usar o código do exemplo 2.

Porém, não vamos sair imprimindo tudo. Devemos fazer um teste antes (**teste condicional IF**) para saber se o número é primo, se for aí sim imprimimos.

Ou seja, vamos ter um IF dentro de um WHILE:

```
numero=1
```

```
max = int(input("Digite um inteiro maior que 1: ") )
```

```
print("Numeros pares entre 1 e", max, ":")
```

```
while numero <= max:
```

```
    if numero%2==0:
```

```
        print(numero, end=" ")
```

```
    numero+=1
```

O end=" " é para mostrar um espaço em branco após imprimir cada número, ao invés de quebra de linha (enter).

```
Digite um inteiro maior que 1: 20
Numeros pares de 1 até 20 :
2 4 6 8 10 12 14 16 18 20
```

- **Exemplo 5** de uso do laço **WHILE**

"Escreva um programa que pede a senha ao usuário, e só sai do looping quando digitarem corretamente a senha"

Obviamente, a senha é o número mais importante do universo:

```
senha='2112'
```

A senha que o usuário vai digitar, vai ficar na variável 'tentativa'.

Se 'senha' e 'tentativa' forem diferentes, o laço while fica repetindo e repetindo e repetindo...sempre pedindo a senha ao usuário.

O laço while só acaba quando as variáveis 'senha' e 'tentativa' forem iguais, ou seja, quando o usuário digitar a senha correta:

```
senha='2112'
tentativa=input("Digite a senha:")

while (senha != tentativa):
    print("Senha errada! Tente novamente!")
    tentativa=input("Digite a senha:")

print("Senha correta. Entrando no sistema...")
```

Pelo amor de deus, agora que sabe como funciona, não vai sair por aí usando seus conhecimentos pra invadir o sistema da Nasa, FBI, Polícia Federal

- **Exemplo 6** de uso do laço **WHILE**

"Programe um script em Python que calcule a média de uma turma, não importa o número de alunos que ela tenha, seu único script serve para todos os casos"

Esse é o problema que descrevemos quando falamos de laços e loopings em Python.

Primeiro, perguntamos quantos alunos tem na turma e armazenamos em "alunos".

Vamos usar uma variável de apoio, chamada 'count', ela vai de 1 até o número de alunos, ok ? Ela que é incrementada no laço while (cresce de um em um).

A soma de todos os alunos, armazenamos na variável 'soma'.

Agora vem o laço while.

O teste é se 'count' é menor que 'alunos'. Se sim, vamos pedir a nota de cada aluno, do primeiro até o aluno de número 'alunos'.

A nota que o usuário digitar, vai ser armazenada na variável 'nota'.

Em seguida, armazenamos na variável 'soma'.

E assim vai indo, de aluno em aluno...no final, temos a soma total das notas de toodo mundo, e não importa o número de alunos na turma, seja um aluno, 10, 20 ou mil alunos, a variável 'soma' vai ter a soma da nota de todo mundo.

Quando acabar o **while** basta dividir essa 'soma' pelo número de alunos ('alunos') e prontinho, temos a média da turma, não importando o tamanho da turma:

```
alunos=int(input("Numeros de alunos na turma: "))
```

```
count=1; soma = 0.0
```

```
while count <= alunos:
```

```
    print("Nota do aluno ", count, ":")
```

```
    nota = float( input() )
```

```
    soma += nota
```

```
    count += 1
```

```
print("Media da turma: ", (soma/alunos) )
```

Foda, essa instrução **while**, não é?

Laço FOR em Python - Um loop controlado

Sem mais delongas, vamos mostrar como é a declaração e estrutura de um laço **for**, em Python:

```
for variavel in [val1, val2, val3, etc]:  
    codigo  
    codigo  
    codigo
```

Funciona basicamente assim.

A variável 'variavel' vai assumir o valor da variável 'val1', e executa o código.

Em seguida, 'variavel' recebe o valor de val2, e executa o código. E assim sucessivamente, até 'variavel' ter recebido todos os valores da sequência.

Ou seja, nosso laço **for** tem três coisas:

A variável que vai receber os valores de uma sequência de dados

A sequência de dados

Código a ser executado em cada iteração

Diferente do loop WHILE, que apenas faz um teste (se for verdadeiro executa um código, se for falso, termina o loop), o loop FOR é dito *controlado*, pois vai executar um determinado número de vezes, que nós escolhemos previamente.

Vamos ver alguns exemplos de uso do laço FOR para firmar os conceitos, só ler essa abstração é meio...abstrato.

- **Exemplo 1** de uso do laço **FOR** em **Python**

"Crie um script em Python que imprima os números de 1 até 5 na tela."

Nossa sequência é: 1, 2, 3, 4, 5

Então, nosso código fica:

```
for val in [1,2,3,4,5]:  
    print(val)
```

Sim, é só isso.

E mais simples é o funcionamento: a variável 'val' recebe o valor 1, e imprime. Depois recebe o valor 2, e imprime....por fim, recebe o valor 5 e imprime.

Como a variável percorreu todos os elementos da lista (esses valores entre colchetes se chamam **list** e estudaremos as listas com afinco, mais na frente em nosso curso), o laço **for** acaba.

- **Exemplo 2** de uso do laço **FOR** em **Python**

"Crie um script que imprime todos os dias da semana na tela."

Não é obrigatório usarmos números na sequência. Uma lista pode ser formada de strings, por exemplo.

Nosso código fica:

```
for dia in ['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta', 'Sábado', 'Domingo']:  
    print(dia)
```

Ou seja, a variável 'dia' recebe o valor de cada string e imprime na tela.

Exercício:

"Escreva um script que imprima os números de 1 até 1000."

*"Nossa senhora, vou ter que usar a sequência [1,2,3,4,5,6,7,8,9...,1000] ???
Deus me livre!"*

Calma, nobre cavaleiro Python. A programação veio para facilitar nossas vidas, e não o contrário.

Não precisa digitar os mil números, é para isso que serve o Python: trabalhar pra gente!

No próximo tutorial de Python, vamos aprender a usar a função range() do Python, que resolve esse problema e muitos outros.

A função range() - O que é, Para que serve e Como usar

Este tutorial é a continuação da aula sobre o laço FOR em Python (clique neste link anterior caso não tenha estudado a aula anterior).

Agora vamos aprender como deixar nossos laços e loopings bem mais poderosos e interessantes, através da função **range**

Função range() do Python

A função range é bem simples e sua funcionalidade é simplesmente nos retornar uma lista de números inteiros.

A maneira mais simples de usar a função range é:
`range(n)`

Onde 'n' é um número.

Ele vai fornecer uma lista com os elementos: 0, 1, 2, ..., (n-1)

Note que ele fornece uma lista de 'n' elementos, começando **sempre do 0**.

Como vai fornecer 'n' elementos e começa do 0, o último elemento é sempre **n-1**

Vamos ver um exemplo, refazendo o exercício 1 da aula passada:

Exemplo 1: Função range em Python

"Crie um script em Python que imprima os números de 1 até 5 na tela, usando a função range."

Nosso código fica assim:

```
for val in range(5):  
    print(val+1)
```

Quando fizemos 'range(5)', essa função vai retornar a lista: [0,1,2,3,4]

Mas queremos números de 1 até 5, então na hora de imprimir basta somar 1 a variável 'val'.

Bem melhor que fazer [1,2,3,4,5], não é verdade?

- **Início e Fim na range()**

No exemplo 1, usamos apenas um argumento (depois você vai aprender melhor o que é um argumento), no caso, um número inteiro.

Porém, podemos usar dois argumentos na range, da seguinte forma:

```
range(inicio, fim)
```

Isso vai nos fornecer uma lista de números, porém ao invés de começar em 0, começa em 'inicio' e vai até 'fim-1'

- **Exemplo 2: Função range em Python**

"Crie um script em Python que imprima os números de 1 até 5 na tela, usando a função range como elemento de início e o de fim."

```
for val in range(1,6):  
    print(val)
```

Note que não precisamos mais imprimir 'val+1', pois a variável 'val' irá receber os valores 1, 2, 3, 4, e o 5 (ou seja, até 6-1)

- **Início, Fim e o Pulo na range()**

Note outra coisa em nossa amada função **range**, ele vai avançando de 1 em 1.

Mas se eu quiser avançar de 2 em 2, 3 em 3 ?

A range nos permite fazer isso, basta dizer o valor desse salto, desse pulo:

```
range(inicio, fim, pulo)
```

- **Exemplo 3: Função range em Python**

"Crie um script em Python que imprima os números pares de 1 até 100."

Os números pares são 2, 4, 6, 8, ..., 100
Eles vão subindo, pulando de 2 em 2.

Veja como fica nosso laço FOR usando a range com pulo:

```
for val in range(2,101,2):  
    print(val)
```

Primeiro a variável recebe par número 2. Depois imprime.
Em seguida, pula pro próximo elemento: $2+2=4$ e então imprime.
Depois, pula pro próximo elemento: $4 + 2 = 6$ e imprime.
Depois, $6+2 = 8$ e imprime...

Atééééé $101-1=100$

Beeeeem melhor usar o laço FOR e a range, não é verdade?
Imagina fazer isso na mão ([2,4,6,8,...,100]), não tem como!

- **Exemplo 4: Tabuada em Python**

"Crie um script em Python que pede qual tabuada o usuário quer ver, em seguida imprima essa tabuada"

Primeiro, pedimos qual a tabuada o usuário quer (do 1, do 2, do 3...) e armazenamos essa informação na variável 'tabuada'.

A tabuada nada mais é que multiplicar um número por 1, depois por 2, depois por 3, depois por 4....até chegar no 10.

Vamos fazer com que a variável 'var' receba os valores de uma lista que vai de 1 até 10, e em cada iteração do laço FOR, multiplicamos pelo número que o usuário nos forneceu, a variável 'tabuada'.

Nosso código de tabuada fica assim:

```
tabuada = int(input("Qual tabuada fazer: ") )  
  
for var in range(1,11):
```

```
print(tabuada*var)
```

Note que fizemos de 1 até 10, se quiser pode fazer até cem, mil, um milhão...tudo com o mesmo código, praticamente.

Interessante Python, não é?

E em pensar que antigamente era 'castigo' e teve de casa, fazer tabuadas gigantescas, agora com 3 linhazinhas de código podemos fazer qualquer tabuada de maneira praticamente instantânea.

Vamos aprender a usar o laço FOR junto da função range para criar uma Progressão Aritmética (PA) em Python.

Progressão Aritmética (PA) em Python

Agora que você estudou aprendeu bem a usar o **laço FOR** e a **função range**, vamos colocar nosso conhecimento em prática e criar qualquer Progressão Aritmética (PA) que o usuário desejar.

- **Progressão Aritmética**

Uma progressão aritmética, também conhecida por PA, nada mais é que uma sequência de números que obedecem a um determinado padrão.

Esse padrão é composto por duas coisas:

Um número inicial

Um número chamado de razão

A sequência começa com o número inicial.

O termo seguinte da sequência é o anterior, somado a razão. E assim sucessivamente.

Vamos supor que temos o termo inicial:

$$a_1 = 1$$

E razão:

$$r = 3$$

A sequência (PA) é:

$$a_1 = 1$$

$$a_2 = a_1 + r = 1 + 3 = 4$$

$$a_3 = a_2 + r = 4 + 3 = 7$$

$$a_4 = a_3 + r = 7 + 3 = 10$$

...

O termo enésimo é descoberto usando a seguinte fórmula:

$$a_n = a_1 + (n - 1) \cdot r$$

$a_n \rightarrow$ termo geral

$a_1 \rightarrow$ 1º termo

$n \rightarrow$ posição do termo geral

$r \rightarrow$ razão da P.A.

Progressão Aritmética (PA) em Python

Crie um programa que pergunte ao usuário o termo inicial e a razão de uma PA.

Em seguida, pergunte a ele quantos elementos da PA ele deseja que seja impresso, e imprima todos os elementos dessa progressão Aritmética, do primeiro termo até o termo 'n' escolhido pelo usuário.

Vamos usar o **laço FOR** para exibir esses elementos da PA.

O primeiro elemento, é o termo inicial que o usuário vai preencher e vamos armazenar na variável 'primeiro'.

Para achar o próximo, é só usar a razão (armazenada na variável 'razao') como o 'pulo', da função range.

E o último termo que devemos exibir?

Ele é o termo 'a(n)', que mostramos como calcular na fórmula na imagem ali de cima.

Repetindo a fórmula:

$$a(n) = a(1) + (n-1) \cdot r$$

Em programação Python, escrevemos assim:

$$\text{ultimo} = \text{primeiro} + (n-1) \cdot r$$

Como a função range(inicial, ultimo) nos fornece os elementos de 'inicial' até 'ultimo-1', vamos somar 1 na variável último, para que a range inclua o valor de 'ultimo' e não até 'ultimo-1', fazemos isso assim:

$$\text{ultimo} = \text{ultimo} + 1$$

Assim, nosso código Python que mostra os termos de uma Progressão Aritmética, fica assim:

```
primeiro=int(input("Primeiro elemento: "))  
razao = int(input("Razao: "))
```

```
n = int(input("Quantos elementos exibir: ") )
```

```
ultimo = primeiro + (n-1)*razao
```

```
ultimo = ultimo + 1
```

```
for var in range(primeiro, ultimo, razao):  
    print(var)
```

Operadores de Atribuição: += , -= , *= , /= e %= (Augmented Assignment)

Neste tutorial de nosso **Curso de Python Online Grátis**, vamos falar sobre os operadores de atribuição (+= , -=, *=, /= e %=), também conhecidos por *Augmented Assignment Statement*, e vamos te mostrar porquê programadores tem fama de preguiçosos.

- **Usando Variáveis de Uma Maneira Diferente**

Fazer uma operação Matemática em Python é moleza, inclusive já te ensinamos nesse link aí e durante o curso estamos usando várias vezes as mais diversas operações.

Uma coisa bem simples é fazer uma soma de duas variáveis, e armazenar em outra.

Por exemplo, você armazena o valor da conta de luz na variável **luz** e o valor da conta de água em **agua**, e quer calcular a despesa total numa variável chamada **total** (sim, nós do Python Progressivo somos super originais com nomes de variáveis). O que fazemos?

Ué: **`total = agua + luz`**

Moleza!

Mas faltou algo tão importante quando a conta de água e luz...a **netflix**. E aí, o que fazer?

Ué:

`total = agua + luz + netflix`

Mas peraí, a gente já tinha o valor da soma 'agua + luz', não precisamos calcular isso de novo. Somos programadores, não gastamos processamento à toa.

Então, fazemos o seguinte:

`total = total + netflix`

Estranho, né?

total = total... como assim algo é igual a ela mesmo mais outra coisa?

O 'total' da esquerda quer dizer:
A variável 'total' vai receber...

O 'total' depois do '=' quer dizer:
Valor anterior de 'total'

Ou seja:

$x = x + y$

Significa: o novo valor de x é o valor anterior de x mais y .

- **Incrementando Uma Variável: $x = x + 1$**

Essa ideia de usar o valor da variável antes da nova expressão, que vai atribuir um novo valor a esta mesma variável, é muito útil.

Tão útil que vamos usar direto, a todo instante, em nosso **Curso de Python**.
O exemplo mais útil dela é para **contadores**, variáveis que vão servir só pra gerar uma contagem.

Vamos ver um exemplo real.

- **Exercício de Python Resolvido**

**Crie um programa que peça um número inteiro positivo ao usuário.
Em seguida, o script deve mostrar os números de 0 até o valor que o usuário escolheu. Use WHILE.**

O valor solicitado ao usuário, vai ficar armazenado na variável n .

Em seguida, vamos usar um contador, uma variável **count** que vai de 0 até n , pela fórmula: $count = count + 1$

Ou seja, a cada iteração do **loop WHILE**, ela aumenta de uma unidade e imprime o valor.

Veja o nosso código Python:

```
n=int(input("Digite um inteiro positivo: "))
```

```
count=0
while count <= n:
    print(count)
    count = count + 1
```

- **Operadores de Atribuição em Python**

Bill Gates tem uma frase bem interessante:

"Escolho uma pessoa preguiçosa pra fazer um trabalho pesado.

Principalmente porque uma pessoa preguiçosa vai encontrar uma forma simples de resolver o problema."

Em outras palavras: programador é um bicho preguiçoso, rs.

Está sempre querendo fazer algo da maneira mais rápida e simples possível, o que é algo bom.

Vamos ensinar agora um atalho em Python.

Sabe o: ***count = count + 1*** ?

Pois é, pode ser substituído por: ***count += 1*** !

Teste:

Outros operadores de atribuição:

x = x - y ***x -= y***

x = x / y ***x /= y***

x = x * y ***x *= y***

x = x % y ***x %= y***

Bem simples e bem fácil.

Apenas uma maneira mais rápido, simples e fácil de escrever algo.

Nada de novo. Sigamos nosso curso.

Como calcular Fatorial em Python

- **Fatorial na Matemática**

O fatorial é uma operação matemática, representada pelo símbolo de exclamação: !

A fórmula de fatorial de um número n é:

$$n! = n * (n-1) * (n-2) * \dots * 3 * 2 * 1$$

Ou seja, o valor de $n!$ (leia-se: n fatorial) é o produto de 1, por 2, por 3, ..., por $(n-1)$ e n .

Por exemplo:

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120 \text{ (ou: } 5! = 5 \times 4! = 5 \times 24 = 120)$$

E por aí vai.

O fatorial é muito usado em um ramo da Matemática chamado Análise Combinatória. Muito mesmo. Se você fez ensino médio, certamente sabe o que é um fatorial.

Agora, vamos levar isso pro nosso amado Python.

- **Fatorial com o laço **WHILE** em Python**

Vamos aprender agora como calcular o fatorial de número usando o **laço While em Python**.

Vamos lá, tente o máximo que puder:

Crie um programa em Python que pede um número inteiro ao usuário e calcule seu fatorial.

Vamos pedir um número inteiro e positivo ao usuário, e armazenar na variável *numero*.

Agora precisamos calcular o produto:

$$1 * 2 * 3 * \dots * (numero-1) * numero$$

Vamos armazenar esse produto na variável *resultado*.
Inicialmente, vamos iniciar a variável *resultado* com 1:

resultado = 1

Agora vamos precisar de um contador, vamos usar a variável **count** para ir de 1 até *numero*, a cada iteração do laço WHILE, ela é incrementada.

Então, dentro do laço while, fazemos com que *resultado* seja multiplicado por *count*, pois *count* vai assumir os valores de 1, 2, 3, ..., até *numero*.

Quando chegar em *numero*, nosso looping while deve parar.
Logo, o teste condicional do while deve ser: **count <= numero** , ok?

Veja como ficou nosso script em Python:

```
numero = int(input("Fatorial de: "))
```

```
resultado=1  
count=1
```

```
while count <= numero:  
    resultado *= count  
    count += 1
```

```
print(resultado)
```

Lembrando que:

•**resultado *= count** é o mesmo que: **resultado = resultado * count**

Ou seja, o novo valor de 'resultado' vai ser o valor antigo multiplicado por 'count'.

•Já: **count += 1** significa: **count = count + 1**

Ou seja, estamos incrementando o valor de 'count' em uma unidade, a cada iteração do laço WHILE.

- **Calcular fatorial usando laço FOR em Python**

A lógica deve ser a mesma. Na verdade, tudo que um laço WHILE faz o FOR faz, e vice-versa.

"Ué, então pra que criar os dois?"

Porque tem situações que é melhor usar um e situações que é melhor usar outro.

Neste caso, você vai ver que é melhor usar o FOR.

Vamos repetir a ideia: Precisamos fazer várias multiplicações, do número 1 até o valor inserido pelo usuário, *numero*.

Do intervalo 1 até *numero*.

Isso te lembra algo? Intervalo?

Sim, **range**.

Nossa querida, amada **função range do Python**, que é unha e carne com o **laço FOR**.

Como queremos um intervalo que vai de 1 até *numero*, basta usarmos:
range(1, numero+1)

Veja como nosso código fica bem mais enxuto e fodástico usando o laço FOR:

```
numero = int(input("Fatorial de: "))
```

```
resultado=1
```

```
for n in range(1,numero+1):  
    resultado *= n
```

```
print(resultado)
```

- **Exercício de Python**

Seu professor de Matemática, cansado de ficar fazendo multiplicações para achar o fatorial, te encomendou um script em Python (obviamente pagou, pois você possui um **certificado do curso de Python**).

Porém, ele quer fazer vários e vários cálculos.

Nos exemplos anteriores a gente pede o número uma vez, mostra o fatorial e fecha o script.

Ele não, ele quer um looping infinito, que só acabe quando ele fornecer o número 0 como entrada.

Se garante? Faça aí!

Loops Aninhados (Laço dentro de laço)

Já falamos aqui em nosso curso sobre **Testes Condicionais IF e ELSE aninhados**, inclusive demos alguns exemplos de uso.

Agora voltamos ao assunto, porém vamos usar **WHILE** e **FOR** dentro de outros WHILE e FOR, por um motivo bem direto: aninhamento é absurdamente importante e usado.

Quando você se tornar uma programador profissional (clique para saber **como ganhar dinheiro programando em Python**), vai usar muito isso, muito mesmo.

Se você já entendeu a lógica dos IF e ELSE aninhados, já sabe tranquilamente aninhar, em teoria, os laços.

Exemplo de loopings aninhados

O melhor exemplo de loop dentro de loop, é o nosso calendário/relógio.

O ano tem 12 meses.

Todo santo ano, começa em Janeiro, depois Fevereiro, Março...até Dezembro.

Depois conta-se tudo de novo, e de novo, e de novo...é um looping.

Porém, dentro de cada mês tem os dias do mês.

Começa no dia 1, dia 2, dia 3...até o último dia do mês (que varia).

Mas quando acaba, volta tudo de novo...*looping*.

E dentro do dia temos as horas!

00h...01h...02h...12h...23h...e depois repete, repete.

Dentro de cada hora?

Loop dos minutos.

Dentro de cada minuto?

Loop dos segundos.

Sim, vivemos em loops aninhados.

Vamos colocar isso em prática programando em Python ?

- **Criando um Tabuleiro**

Faça um programa em Python que solicite um número positivo inteiro ao usuário, e depois exiba um tabuleiro na tela, com igual número de linhas e colunas.

Por exemplo, se ele digitar 3 deve aparecer na tela:

```
x x x
x x x
x x x
```

Se digitar 2:

```
x x
x x
```

Se digitar 4:

```
x x x x
x x x x
x x x x
x x x x
```

Solução:

Vamos criar um tabuleiro de tamanho **N x N**, onde esse N o usuário fornece e armazenamos na variável **n**.

Usaremos duas variáveis de controle: **linha** e **coluna**, ok?

O primeiro laço FOR vai ser responsável por percorrer as linhas.

Ele vai percorrer um intervalo de **n** linhas, ou seja: range(n)

Dentro de cada linha dessa, precisamos desenhar as colunas, onde cada coluna printamos um "x ". E quantas vezes isso é feito? **n** vezes.

Ou seja, vamos usar outro intervalo de valor **n**: range(n)

Vamos usar uma variação da função **print**, que não quebra a linha automaticamente. Para isso, basta adicionar o **end=""** na função print, pra definir que no fim dela não fala nada, não coloque nenhum caractere (por padrão ela coloca **\n** de quebra de linha).

Quando terminar de imprimir cada linha, o que fazemos?

Precisamos dar uma quebra de linha, para o tabuleiro ficar bonitinho, então usamos **print()** (isso mesmo, sem nada dentro).

Veja como nosso código ficou:

```
print("Vamos criar um tabuleiro de tamanho: N x N")
n=int(input("Valor de N: "))

for linha in range(n):
    for coluna in range(n):
        print("x ",end="")
    print()
```

Guarde esse código com carinho, vamos usar no futuro quando formos programar diversos jogos, como jogo da velha, batalha naval etc.

- **Exercício resolvido de Python**

Crie um programa que exiba todos os segundos e minutos no intervalo de uma hora, no seguinte formato: XXmin YYs

Por exemplo:

12min59s

Vamos usar duas variáveis, a **minuto** e **segundo**.

Ambas vão de 0 até 59, ou seja, vamos usar o intervalo de 60: **range(60)**

O primeiro FOR é para os minutos.

Primeiro o minuto 0. Dentro dele vamos imprimir do 0s até 59s.

Depois o minuto 1. Dentro deles imprimimos de 0s até 59s.

...

Por fim, minuto 59.

O FOR que vai estar aninhado é o responsável por imprimir os segundos.

Veja como ficou o código do programa:

```
for minutos in range(60):
    for segundos in range(60):
        print(minutos,"min",segundos,"s")
```

- **Exercício Resolvido de Python**

Faça o mesmo do exemplo acima, mas adicione as horas para ficar no formato: XXh YYmin ZZs

Aqui, basta notar que as horas vão de 0h até 23h: `range(24)`

```
for horas in range(24):
    for minutos in range(60):
        for segundos in range(60):
            print(horas,"h",minutos,"min",segundos,"s")
```

(se demorar muito, e vai demorar, dê um control+C que o shell do Python vai parar de rodar)

- **Exercício Resolvido de Laço Aninhado**

Você foi contratado por uma escola para fazer o seguinte script em Python: Primeiro, pergunta a quem vai usar o script quantos alunos tem na sala. Depois, quantas matérias cada aluno estuda.

Em seguida o usuário vai preenchendo a nota de cada matéria, de cada aluno.

Seu programa deve fornecer a média de cada aluno e a média geral da turma.

- **Solução:**

Vamos armazenar o número de alunos na variável **alunos** e o número de matérias em **materias**.

O primeiro laço é o que vai percorrer os alunos:

Aluno 1, depois aluno 2, o aluno 3...ou seja: **range(alunos)**

Depois, para cada aluno vamos dar as respectivas notas de cada matéria.

A cada nota que o usuário der, vamos somar ela na variável '**mediaAluno**', e ao final basta dividirmos esse número pelo número de matérias e temos a média daquele aluno.

A cada iteração do primeiro FOR, antes de digitarmos as notas de cada matéria, devemos zerar essa variável!

Quando tivermos a '**mediaAluno**', vamos somar esse valor na variável '**mediaTurma**', e no final do programa dividimos essa variável pelo número de alunos, para ter a média geral da turma.

Veja como ficou o código final:

```
alunos=int(input("Quantos alunos tem na turma: "))
materias=int(input("Quantas matérias eles estudam: "))

mediaTurma=0
for aluno in range(alunos):
    print("Aluno",aluno+1,":")

    mediaAluno=0
    for materia in range(materias):
        print("Nota da materia",materia+1,":", end="")
        nota=int(input())
        mediaAluno += nota

    mediaAluno /= materias
    print("Media desse aluno:",mediaAluno,"\n")

    mediaTurma += mediaAluno

mediaTurma /= alunos
print("Media da turma:",mediaTurma)
```

- **Python na Mega-Sena**

Faça um script em Python que exiba todos os possíveis palpites da Mega-Sena.

Dados:

- 1.Cada palpite possui 6 dezenas
- 2.As dezenas variam de 1 até 60
- 3.Não pode repetir dezena

- Solução

Existem diversos estudos e pesquisas que tentam usar a programação para ajudar, de alguma maneira, as pessoas sobre **como ganhar na Mega-Sena**.

Antes de iniciarmos a parte da programação Python, precisamos definir algumas coisas.

Vamos usar 6 variáveis, para representar as 6 dezenas da Mega-Sena, ok?
Elas serão:

dez1 dez2 dez3 dez4 dez5 dez6

A princípio, você poderia pensar:

Só usar laço FOR dentro de laço FOR, para cada uma das dezenas, variando de 1 até 60.

Parece um raciocínio que faz sentido.

Mas temos um problema: se fizer isso, você vai criar palpites com dezenas repetidas, experimente:

O script acima tem vários e vários palpites com dezenas repetidas:

1 1 1 1 1 1
1 2 3 4 5 5
etc etc

Podemos resolver isso deixando uma coisa bem clara:

O palpite "dez1 dez2 dez3 dez4 dez5 dez6" está em ordem crescente.

Ou seja:

dez1 > dez2 > dez3 > dez4 > dez5 > dez6

Isso quer dizer algumas coisas interessantes.

Por exemplo, a última dezena vai de 6 até 60.

Não tem como a última dezena ser 5, por exemplo.

Pois teríamos o menor palpite: 1 2 3 4 5...e a última dezena tem que ser 6 pra cima.

O grande segredo para esse tipo de questões é fazer com que:

dez2 > dez1
dez3 > dez2
dez4 > dez3
dez5 > dez4
dez6 > dez5

Ou seja, para **dez1**, vamos deixar o intervalo como **range(60)**. Já para **dez2**, o range deve começar de modo a ser sempre maior que **dez1**, e isso é feito assim: **range(dez1+1, 60)**.

A **dez3** deve ser sempre maior que **dez2**, isso é feito usando o seguinte intervalo: **range(dez2+1,60)**, e assim sucessivamente.

Script Que Exibe Todos os Palpites da Mega-Sena

Nosso script fica então:

```
for dez1 in range(60):
    for dez2 in range(dez1+1,60):
        for dez3 in range(dez2+1,60):
            for dez4 in range(dez3+1,60):
                for dez5 in range(dez4+1,60):
                    for dez6 in range(dez5+1,60):
                        print(dez1+1,dez2+1,dez3+1,dez4+1,dez5+1,dez6+1)
```

Se você rodar o script acima, certamente vai levar um bom tempo para ver todos os possíveis palpites, que são mais de 50 milhões.

Ao invés de exibir todos, você pode usar um contador, para contar o número de possíveis palpites, veja:

```
total=0
for dez1 in range(60):
    for dez2 in range(dez1+1,60):
        for dez3 in range(dez2+1,60):
            for dez4 in range(dez3+1,60):
                for dez5 in range(dez4+1,60):
                    for dez6 in range(dez5+1,60):
                        total += 1
print(total)
```

Entrando no site oficial da Meg-Sena:

<http://loterias.caixa.gov.br/wps/portal/loterias/landing/megasena/>

Lá, na seção "Probabilidade", vemos que a chance de uma pessoa ganhar é de 1 em 50.063.860 . Mas, hora, esse é o número que nosso programa calculou.

Ou seja, nosso script está correto.

Instruções ELSE, BREAK e CONTINUE em Laços

- Instrução **ELSE** em **Python**

Já falamos da **instrução ELSE** em **Testes Condicionais do tipo IF ELSE ELIF**. Essa instrução, embora não seja muito famosa, também existe junto com o laço WHILE.

Sua estrutura é a seguinte:

```
while TESTE:
    codigo
    codigo
    ...
else:
    codigo
    ...
```

A lógica do ELSE junto com o WHILE é bem simples: assim que sai do **laço WHILE** (sem ser via instrução BREAK), o código dentro da instrução ELSE será executado.

Ou seja, enquanto o **TESTE** for verdadeiro, ele roda o código dentro do WHILE.

Quando a condição vira falso, e não se usa uma instrução **BREAK**, o código do ELSE será executado (com uma mensagem de saída, de desligamento de sistema, por exemplo).

- Instrução **BREAK** em **Python**

Outro comando muito importante e usado em laços Python, é o **BREAK**, que significa 'quebrar', 'interromper', 'pausar'. E é isso que faz.

```
while TESTE:
    codigo
    if TESTE2:
        break;
    ...
```

A função do BREAK é simplesmente parar o looping.
Se fizer um teste, ele der positivo e você usar a **break**, o laço é automaticamente findado.

- **Instrução CONTINUE em Python**

Na instrução **BREAK**, quando ela é executada, tudo para, tudo acaba, adeus laço, adeus WHILE, adeus FOR, termina ele.

Se ao invés de **break** usar **continue**, o laço não é terminado. Porém, ele pula do **continue** pro início do laço, tudo que tem ali em diante do **continue** não é mais executado.

Exemplo de Código Python

A instrução **pass** é muito utilizada quando queremos um código vazio, uma instrução vazia, sem fazer nada de específico.

O código a seguir é um looping infinito, roda eternamente, esse código só vai parar de executar se você digitar CONTROL + C

```
while 1:  
    pass
```

- **Exercício Resolvido de Python**

Escreva um programa em Python que vai somar todos os números de 1 até 1 milhão, menos os que são múltiplos de 3.

A soma total ficará armazenada em **total**.

Vamos percorrer todos os números de 1 até 1 milhão, quem vai assumir todos esses valores é a variável **count**, nosso contador.

Primeiro incrementamos nosso contador. Ele começa em 0 e logo vira 1.

Agora, vamos fazer um teste para saber se o valor de **count** é múltiplo de 3.

Se for múltiplo de 3, damos uma instrução **continue** e o loop **for** segue para o próximo elemento, nada é feito, pulamos para a próxima iteração.

Caso não seja múltiplo de 3, vamos somar esse valor do contador na variável

total.

Prontinho, nosso programa percorre de 1 até 1 milhão e soma tudo que não for múltiplo de 3:

```
total=0
```

```
for count in range(1000000):  
    count += 1  
    if(count % 3 == 0 ): continue  
    total += count
```

```
print(total)
```

Lista de Exercícios de Laços e Loops

1. Faça um programa que peça uma nota, entre zero e dez. Mostre uma mensagem caso o valor seja inválido e continue pedindo até que o usuário informe um valor válido.
2. Faça um programa que leia um nome de usuário e a sua senha e não aceite a senha igual ao nome do usuário, mostrando uma mensagem de erro e voltando a pedir as informações.
3. Faça um programa que leia e valide as seguintes informações:
Nome: maior que 3 caracteres;
Idade: entre 0 e 150;
Salário: maior que zero;
Sexo: 'f' ou 'm';
Estado Civil: 's', 'c', 'v', 'd';
Use a função **len(string)** para saber o tamanho de um texto (número de caracteres).
4. Supondo que a população de um país A seja da ordem de 80000 habitantes com uma taxa anual de crescimento de 3% e que a população de B seja 200000 habitantes com uma taxa de crescimento de 1.5%. Faça um programa que calcule e escreva o número de anos necessários para que a população do país A ultrapasse ou iguale a população do país B, mantidas as taxas de crescimento.
5. Altere o programa anterior permitindo ao usuário informar as populações e as taxas de crescimento iniciais. Valide a entrada e permita repetir a operação.
6. Faça um programa que imprima na tela os números de 1 a 20, um abaixo do outro. Depois modifique o programa para que ele mostre os números um ao lado do outro.
7. Faça um programa que leia 5 números e informe o maior número.
8. Faça um programa que leia 5 números e informe a soma e a média dos números.

9. Faça um programa que imprima na tela apenas os números ímpares entre 1 e 50.

10. Faça um programa que receba dois números inteiros e gere os números inteiros que estão no intervalo compreendido por eles.

11. Altere o programa anterior para mostrar no final a soma dos números.

12. Desenvolva um gerador de tabuada, capaz de gerar a tabuada de qualquer número inteiro entre 1 a 10. O usuário deve informar de qual numero ele deseja ver a tabuada. A saída deve ser conforme o exemplo abaixo:

Tabuada de 5:

$5 \times 1 = 5$

$5 \times 2 = 10$

...

$5 \times 10 = 50$

13. Faça um programa que peça dois números, base e expoente, calcule e mostre o primeiro número elevado ao segundo número. Não utilize a função de potência da linguagem.

14. Faça um programa que peça 10 números inteiros, calcule e mostre a quantidade de números pares e a quantidade de números ímpares.

A série de Fibonacci é formada pela sequência 1,1,2,3,5,8,13,21,34,55,...

15. Faça um programa capaz de gerar a série até o n-ésimo termo.

16. A série de Fibonacci é formada pela sequência

0,1,1,2,3,5,8,13,21,34,55,... Faça um programa que gere a série até que o valor seja maior que 500.

17. Faça um programa que calcule o fatorial de um número inteiro fornecido pelo usuário. Ex.: $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$

18. Faça um programa que, dado um conjunto de N números, determine o menor valor, o maior valor e a soma dos valores.

19. Altere o programa anterior para que ele aceite apenas números entre 0 e 1000.
20. Altere o programa de cálculo do fatorial, permitindo ao usuário calcular o fatorial várias vezes e limitando o fatorial a números inteiros positivos e menores que 16.
21. Faça um programa que peça um número inteiro e determine se ele é ou não um número primo. Um número primo é aquele que é divisível somente por ele mesmo e por 1.
22. Altere o programa de cálculo dos números primos, informando, caso o número não seja primo, por quais número ele é divisível.
23. Faça um programa que mostre todos os primos entre 1 e N sendo N um número inteiro fornecido pelo usuário. O programa deverá mostrar também o número de divisões que ele executou para encontrar os números primos. Serão avaliados o funcionamento, o estilo e o número de testes (divisões) executados.
24. Faça um programa que calcule o mostre a média aritmética de N notas.
25. Faça um programa que peça para n pessoas a sua idade, ao final o programa devera verificar se a média de idade da turma varia entre 0 e 25,26 e 60 e maior que 60; e então, dizer se a turma é jovem, adulta ou idosa, conforme a média calculada.
26. Numa eleição existem três candidatos. Faça um programa que peça o número total de eleitores. Peça para cada eleitor votar e ao final mostrar o número de votos de cada candidato.
27. Faça um programa que calcule o número médio de alunos por turma. Para isto, peça a quantidade de turmas e a quantidade de alunos para cada turma. As turmas não podem ter mais de 40 alunos.
28. Faça um programa que calcule o valor total investido por um colecionador em sua coleção de CDs e o valor médio gasto em cada um deles. O usuário deverá informar a quantidade de CDs e o valor para em cada um.

29. O Sr. Manoel Joaquim possui uma grande loja de artigos de R\$ 1,99, com cerca de 10 caixas. Para agilizar o cálculo de quanto cada cliente deve pagar ele desenvolveu um tabela que contém o número de itens que o cliente comprou e ao lado o valor da conta. Desta forma a atendente do caixa precisa apenas contar quantos itens o cliente está levando e olhar na tabela de preços. Você foi contratado para desenvolver o programa que monta esta tabela de preços, que conterá os preços de 1 até 50 produtos, conforme o exemplo abaixo:

Lojas Quase Dois - Tabela de preços

1 - R\$ 1.99

2 - R\$ 3.98

...

50 - R\$ 99.50

30. O Sr. Manoel Joaquim acaba de adquirir uma panificadora e pretende implantar a metodologia da tabelinha, que já é um sucesso na sua loja de 1,99. Você foi contratado para desenvolver o programa que monta a tabela de preços de pães, de 1 até 50 pães, a partir do preço do pão informado pelo usuário, conforme o exemplo abaixo:

Preço do pão: R\$ 0.18

Panificadora Pão de Ontem - Tabela de preços

1 - R\$ 0.18

2 - R\$ 0.36

...

50 - R\$ 9.00

31. O Sr. Manoel Joaquim expandiu seus negócios para além dos negócios de 1,99 e agora possui uma loja de conveniências. Faça um programa que implemente uma caixa registradora rudimentar. O programa deverá receber um número desconhecido de valores referentes aos preços das mercadorias. Um valor zero deve ser informado pelo operador para indicar o final da compra. O programa deve então mostrar o total da compra e perguntar o valor em dinheiro que o cliente forneceu, para então calcular e mostrar o valor do troco. Após esta operação, o programa deverá voltar ao ponto inicial, para registrar a próxima compra. A saída deve ser conforme o exemplo abaixo:

Lojas Tabajara

Produto 1: R\$ 2.20

Produto 2: R\$ 5.80

Produto 3: R\$ 0

Total: R\$ 9.00

Dinheiro: R\$ 20.00

Troco: R\$ 11.00

...

32. Faça um programa que calcule o fatorial de um número inteiro fornecido pelo usuário. Ex.: $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$. A saída deve ser conforme o exemplo abaixo:

Fatorial de: 5

$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$

33. O Departamento Estadual de Meteorologia lhe contratou para desenvolver um programa que leia as um conjunto indeterminado de temperaturas, e informe ao final a menor e a maior temperaturas informadas, bem como a média das temperaturas.

34. Os números primos possuem várias aplicações dentro da Computação, por exemplo na Criptografia. Um número primo é aquele que é divisível apenas por um e por ele mesmo. Faça um programa que peça um número inteiro e determine se ele é ou não um número primo.

35. Encontrar números primos é uma tarefa difícil. Faça um programa que gera uma lista dos números primos existentes entre 1 e um número inteiro informado pelo usuário.

36. Desenvolva um programa que faça a tabuada de um número qualquer inteiro que será digitado pelo usuário, mas a tabuada não deve necessariamente iniciar em 1 e terminar em 10, o valor inicial e final devem ser informados também pelo usuário, conforme exemplo abaixo:

Montar a tabuada de: 5

Começar por: 4

Terminar em: 7

Vou montar a tabuada de 5 começando em 4 e terminando em 7:

$$5 \times 4 = 20$$

$$5 \times 5 = 25$$

$$5 \times 6 = 30$$

$$5 \times 7 = 35$$

Obs: Você deve verificar se o usuário não digitou o final menor que o inicial.

37. Uma academia deseja fazer um senso entre seus clientes para descobrir o mais alto, o mais baixo, a mais gordo e o mais magro, para isto você deve fazer um programa que pergunte a cada um dos clientes da academia seu código, sua altura e seu peso. O final da digitação de dados deve ser dada quando o usuário digitar 0 (zero) no campo código. Ao encerrar o programa também deve ser informados os códigos e valores do cliente mais alto, do mais baixo, do mais gordo e do mais magro, além da média das alturas e dos pesos dos clientes

38. Um funcionário de uma empresa recebe aumento salarial anualmente: Sabe-se que:

Esse funcionário foi contratado em 1995, com salário inicial de R\$ 1.000,00;

Em 1996 recebeu aumento de 1,5% sobre seu salário inicial;

A partir de 1997 (inclusive), os aumentos salariais sempre correspondem ao dobro do percentual do ano anterior. Faça um programa que determine o salário atual desse funcionário. Após concluir isto, altere o programa permitindo que o usuário digite o salário inicial do funcionário.

39. Faça um programa que leia dez conjuntos de dois valores, o primeiro representando o número do aluno e o segundo representando a sua altura em centímetros. Encontre o aluno mais alto e o mais baixo. Mostre o número do aluno mais alto e o número do aluno mais baixo, junto com suas alturas.

40. Foi feita uma estatística em cinco cidades brasileiras para coletar dados sobre acidentes de trânsito. Foram obtidos os seguintes dados:

Código da cidade;

Número de veículos de passeio (em 1999);

Número de acidentes de trânsito com vítimas (em 1999). Deseja-se saber:

Qual o maior e menor índice de acidentes de transito e a que cidade pertence;

Qual a média de veículos nas cinco cidades juntas;

Qual a média de acidentes de trânsito nas cidades com menos de 2.000 veículos de passeio.

41. Faça um programa que receba o valor de uma dívida e mostre uma tabela com os seguintes dados: valor da dívida, valor dos juros, quantidade de parcelas e valor da parcela.

Os juros e a quantidade de parcelas seguem a tabela abaixo:

Quantidade de Parcelas % de Juros sobre o valor inicial da dívida

1	0
3	10
6	15
9	20
12	25

Exemplo de saída do programa:

Valor da Dívida Valor dos Juros Quantidade de Parcelas Valor da Parcela

R\$ 1.000,00	0	1	R\$ 1.000,00
R\$ 1.100,00	100	3	R\$ 366,00
R\$ 1.150,00	150	6	R\$ 191,67

42. Faça um programa que leia uma quantidade indeterminada de números positivos e conte quantos deles estão nos seguintes intervalos: [0-25], [26-50], [51-75] e [76-100]. A entrada de dados deverá terminar quando for lido um número negativo.

O cardápio de uma lanchonete é o seguinte:

Especificação Código Preço

Cachorro Quente	100	R\$ 1,20
Bauru Simples	101	R\$ 1,30
Bauru com ovo	102	R\$ 1,50
Hambúrguer	103	R\$ 1,20
Cheeseburger	104	R\$ 1,30
Refrigerante	105	R\$ 1,00

43. Faça um programa que leia o código dos itens pedidos e as quantidades desejadas. Calcule e mostre o valor a ser pago por item (preço * quantidade)

e o total geral do pedido. Considere que o cliente deve informar quando o pedido deve ser encerrado.

44. Em uma eleição presidencial existem quatro candidatos. Os votos são informados por meio de código. Os códigos utilizados são:

1 , 2, 3, 4 - Votos para os respectivos candidatos

(você deve montar a tabela ex: 1 - Jose/ 2- João/etc)

5 - Voto Nulo

6 - Voto em Branco

Faça um programa que calcule e mostre:

O total de votos para cada candidato;

O total de votos nulos;

O total de votos em branco;

A percentagem de votos nulos sobre o total de votos;

A percentagem de votos em branco sobre o total de votos. Para finalizar o conjunto de votos tem-se o valor zero.

45. Desenvolver um programa para verificar a nota do aluno em uma prova com 10 questões, o programa deve perguntar ao aluno a resposta de cada questão e ao final comparar com o gabarito da prova e assim calcular o total de acertos e a nota (atribuir 1 ponto por resposta certa). Após cada aluno utilizar o sistema deve ser feita uma pergunta se outro aluno vai utilizar o sistema. Após todos os alunos terem respondido informar:

Maior e Menor Acerto;

Total de Alunos que utilizaram o sistema;

A Média das Notas da Turma.

Gabarito da Prova:

01 - A

02 - B

03 - C

04 - D

05 - E

06 - E

07 - D

08 - C

09 - B

10 - A

Após concluir isto você poderia incrementar o programa permitindo que o professor digite o gabarito da prova antes dos alunos usarem o programa.

46. Em uma competição de salto em distância cada atleta tem direito a cinco saltos. No final da série de saltos de cada atleta, o melhor e o pior resultados são eliminados. O seu resultado fica sendo a média dos três valores restantes. Você deve fazer um programa que receba o nome e as cinco distâncias alcançadas pelo atleta em seus saltos e depois informe a média dos saltos conforme a descrição acima informada (retirar o melhor e o pior salto e depois calcular a média). Faça uso de uma lista para armazenar os saltos. Os saltos são informados na ordem da execução, portanto não são ordenados. O programa deve ser encerrado quando não for informado o nome do atleta. A saída do programa deve ser conforme o exemplo abaixo:
Atleta: Rodrigo Curvêllo

Primeiro Salto: 6.5 m
Segundo Salto: 6.1 m
Terceiro Salto: 6.2 m
Quarto Salto: 5.4 m
Quinto Salto: 5.3 m

Melhor salto: 6.5 m
Pior salto: 5.3 m
Média dos demais saltos: 5.9 m

Resultado final:
Rodrigo Curvêllo: 5.9 m

47. Em uma competição de ginástica, cada atleta recebe votos de sete jurados. A melhor e a pior nota são eliminadas. A sua nota fica sendo a média dos votos restantes. Você deve fazer um programa que receba o nome do ginasta e as notas dos sete jurados alcançadas pelo atleta em sua apresentação e depois informe a sua média, conforme a descrição acima informada (retirar o melhor e o pior salto e depois calcular a média com as notas restantes). As notas não são informados ordenadas. Um exemplo de saída do programa deve ser conforme o exemplo abaixo:

Atleta: Aparecido Parente
Nota: 9.9
Nota: 7.5
Nota: 9.5
Nota: 8.5

Nota: 9.0

Nota: 8.5

Nota: 9.7

Resultado final:

Atleta: Aparecido Parente

Melhor nota: 9.9

Pior nota: 7.5

Média: 9,04

48. Faça um programa que peça um numero inteiro positivo e em seguida mostre este numero invertido.

Exemplo:

12376489

=> 98467321

49. Faça um programa que mostre os n termos da Série a seguir:

$$S = 1/1 + 2/3 + 3/5 + 4/7 + 5/9 + \dots + n/m.$$

Imprima no final a soma da série.

50. Sendo $H = 1 + 1/2 + 1/3 + 1/4 + \dots + 1/N$, Faça um programa que calcule o valor de H com N termos.

51. Faça um programa que mostre os n termos da Série a seguir:

$$S = 1/1 + 2/3 + 3/5 + 4/7 + 5/9 + \dots + n/m.$$

Imprima no final a soma da série.

- **Soluções**

1.

Como o nome diz, validar um dado é verificar, checar, garantir que aquelas informações são importantes, se servem de algo.

Se um código Python tá esperando um número e recebe a informação 'batata' do usuário, vai dar erro, vai sair coisa errada daí.

Já viu aqueles exemplos:

Digite sua data (dd/mm/aaaa), em algum formulário?

Pois é, é um tipo de validação, ele quer que o usuário digite o dia com 2 dígitos, o mês com 2 dígitos e o ano de nascimento com 4 dígitos.

Aí vai o usuário e digita '9/7/99' ... um bom código bem validado simplesmente **não aceita** essa informação. E é isso que vamos ver com alguns exemplos de código.

Vamos aprender a validar simplesmente barrando alguma asneira que o usuário vai fazer, ok?

Temos que criar um script que fique sempre repetindo, pedindo a nota correta, caso ela seja errada. Quantas vezes ? Pra sempre?
Ou seja...pede...testa...não validou? Pede de novo...testa...não validou?
Roda de novo.

Isso te lembra algo? Sim, **laço WHILE**, ele que teste algo e roda enquanto o teste for verdadeiro!

Nosso código fica:

```
nota = float(input("Insira uma nota 0 até 10: "))
```

```
while (nota < 0) or (nota > 10):
```

```
    nota = float(input("Não pode ser menor que 0 ou maior que 10 meu jovem!\n"))
```

```
    Tente novamente:"))
```

```
print("Nota válida")
```

O *looping* WHILE é teimoso, só sai dela se a notar for de 0 até 10. Mais ou menos, ele não sai. Pode digitar até -0.1 ou 10.0001 (nunca duvide da capacidade de um usuário), ele não sai, pede a nota de novo!

2.

A lógica é a mesma da questão anterior, porém, aqui, a gente vai comparar *strings* (vamos estudar elas mais a fundo em breve). Bacana que essa lógica é bem usada na vida real.

Vários sites, programas etc, não aceitam você ter mesmo login e senha. Veja que no decorrer de nosso Curso de Python, nossos programas vão se tornando cada vez mais realistas:

```
login = input("Login: ")
senha = input("Senha: ")
```

```
while login == senha:
    print("Sua senha deve ser diferente do login: ")
    senha = input("Senha: ")
```

```
print("Cadastro aprovado")
```

3.

Inicialmente, pedimos as 5 informações ao usuário (sempre informe como deve ser essa informação que ele deve fornecer, o seu formato, para ele informar corretamente).

Depois, vamos usando 5 loopings WHILE para nos certificar se ele forneceu os dados corretos. Vejam como ficou nosso script em Python:

```
nome = input("Qual seu nome [mínimo 4 caracteres]: ")
idade = int(input("Sua idade: "))
salario = float(input("Salário: "))
sexo = input("Sexo ('f' para feminino ou 'm' para masculino): ")
civil = input("Estado civil (s, c, v ou d): ")
```

```
while len(nome) <= 3:
    nome = input("Seu nome deve ter mais que 3 caracteres: ")
```

```
while (idade < 0) or (idade > 150):
    idade = int(input("Você deve ter entre 0 e 150 anos: "))
```

```
while (salario < 0):
    salario = float(input("A coisa tá difícil, mas não tem salário negativo: "))
```

```
while (sexo != 'f') and (sexo != 'm'):
    sexo = input("Biologicamente, você deve ser 'f' ou 'm': ")
```

```
while (civil!='s')and(civil!='c')and(civil!='v')and(civil!='d'):
    print("Nao tem estado civil 'confuso'")
    civil = input("Deve ser s, c, v ou d: ")
```

4. e 5.

Precisamos entender uma coisa:

O país A só vai superar o país B em termos de população se a taxa de crescimento de A for superior a de B.

Então, se assume que:

População de A é menor que B

Taxa de crescimento de A é maior que B

Então, nosso código fica assim:

```
popA=int(input("População do país A: "))
while popA<0:
    popA=int(input("População do país deve ser maior que 0: "))
taxaA=float(input("Taxa de crescimento da cidade A: "))

popB=int(input("População do país B: "))
while popB<0:
    popB=int(input("População do país deve ser maior que 0: "))
taxaB=float(input("Taxa de crescimento da cidade B: "))

ano=0
while popA < popB:
    ano += 1
    popA = int((1 + (taxaA/100) )* popA)
    popB = int((1 + (taxaB/100) )* popB)
    print("Ano %d:" % ano)
    print("Populacao A: %d" % popA)
    print("População B: %d\n\n" % popB)

print("Ultrapassa no ano:",ano)
```

Crie agora um script que não importa o tamanho ou taxa de crescimento, diga o ano que uma supera a outra cidade ou que isso nunca vai acontecer.

7. e 8.

Primeiro, perguntamos quantos números o usuário que inserir e armazenamos em '**numeros**'.

O primeiro número fornecido é especial, digamos assim.
Como só foi fornecido ele, podemos dizer que é o maior até então.

O maior número é 'primeiro', bem como a soma total dos números digitados, 'soma', também é 'primeiro'.

Vamos usar uma variável de controle que vai de 1 até 'numeros', é a 'count'.

Os dados vão ser fornecidos dentro do loop WHILE.
Cada vez que alguém insere um novo número, na variável 'temp', eu checo se esse número é maior que 'maior', meu maior número armazenado.

Se for, o novo valor de 'temp' é esse número. Se não for, nada ocorre.

Cada vez que insiro esse novo número em 'temp', somo esse valor na soma total 'soma'.

Veja o código de nosso script Python:

```
numeros=int(input("Quantos numeros: "))

primeiro=int(input("Numero 1: "))

count=1
maior=primeiro
soma=primeiro

while count< numeros:
    count += 1
    temp=int(input("Numero %d: " % count))
    soma += temp
    if temp>maior:
        maior = temp

media = soma / numeros
print("Soma:",soma)
print("Maior:", maior)
print("Media: %.2f" % (soma/numeros))
```

12.

Primeiro, perguntamos qual tabuada o usuário quer: a de 1, de 2, de 3 ou qualquer outra, e armazenamos na variável 'tabuada'.

Em seguida, usamos uma variável **count** que vai de 1 até 10 dentro do laço FOR e função range, veja:

```
tabuada=int(input("Tabuada do numero: "))
```

```
for count in range(10):  
    print("%d x %d = %d" % (tabuada, count+1, tabuada*(count+1)) )
```

13.

Bom, vamos lá.

Precisamos garantir duas coisas: que a base e o expoente sejam números positivos, e o expoente seja também inteiro.

Vamos armazenar o valor da base na variável **base** e o expoente em **expoente**.

Nosso resultado vai ficar armazenado na variável **potencia**, que deve iniciar com valor 1.

Temos que multiplicar a variável **potencia** por **base**, várias vezes. Quantas vezes? **expoente** vezes!

Vamos usar uma variável de apoio **count** que serve de contador de 1 até **potencia**, que vai controlar quantas vezes os laços serão executados.

Exercício resolvido usando laço WHILE

```
print("base ^ expoente:")  
base=int(input("Base: "))  
expoente=int(input("Expoente: "))
```

```
potencia=1  
count=1
```

```
while count <= expoente:  
    potencia *= base  
    count +=1
```

```
print(base,"^",expoente,"=",potencia)
```

Exercício resolvido usando laço FOR

```
print("base ^ expoente:")
base=int(input("Base: "))
expoente=int(input("Expoente: "))

potencia=1

for count in range(expoente):
    potencia *= base
    count += 1

print(base,"^",expoente,"=",potencia)
```

15.

Antes de nossa lógica funcionar no código, ela precisa funcionar em nossa cabeça. Esse é o grande segredo de todo bom programador: a coisa precisa funcionar na sua mente antes de você começar a *codar*, ok?

Vamos chamar de **n** a variável que representa o n-ésimo termo da série, termo que o usuário vai fornecer.

Como temos sempre que somar os dois elementos anteriores, as variáveis para representar eles terão os nomes **ultimo** e **penultimo**. O n-ésimo termo vai ser armazenado na variável **termo**.

Agora vamos, com calma, te ensinar o pulo do gato.

Vamos pegar um situação que temos a seguinte configuração:

```
ultimo = 5
penultimo = 3
```

Logo, o próximo termo é:
`termo = ultimo + penultimo = 8`

Depois, na nova iteração do laço, nossos novos valores de **ultimo** e **penultimo** devem variar, ir pra frente na sequência de Fibonacci. Devem ser agora:

```
ultimo = 8
penultimo = 5
```

Ou seja, o novo valor de **penultimo** deve ser o valor antigo de **ultimo**:
`penultimo = ultimo`

E o novo valor de **ultimo** deve ser o próximo termo da sequência.
E como calculamos o próximo valor da sequência? Somando os dois valores anteriores, que é o valor antigo de **ultimo** com **penultimo**.

O novo valor de **ultimo** fica:
 $\text{ultimo} = \text{ultimo} + \text{penultimo}$

Ou simplesmente:
 $\text{ultimo} = \text{termo}$

Exercício resolvido com laço WHILE

Lembrando que se o usuário pedir o termo de número 1 ou 2, devemos retornar 1. Checamos isso com um **IF** retornando 1 caso o usuário digite 1 ou 2.

Só daí em diante que aplicamos a regrinha da soma dos valores anteriores, aí cai no **ELSE**, onde iremos fazer o código explicado pela lógica anterior.

A variável que vai ficar responsável pela contagem de iterações é a **count**. Ela se inicia valendo 3, pois já temos os dois valores iniciais da sequência (1 e 1) e vamos calcular em seguida o terceiro termo.

Quando o laço WHILE acaba, damos o print da variável **termo**.
(tudo ainda dentro do **ELSE**)

Nosso código fica assim:

```
n = int(input("Que termo deseja encontrar: "))
ultimo=1
penultimo=1
```

```
if (n==1) or (n==2):
    print("1")
else:
    count=3
    while count <= n:
        termo = ultimo + penultimo
        penultimo = ultimo
        ultimo = termo
        count += 1
    print(termo)
```

Exercício resolvido com laço FOR

```
n = int(input("Que termo deseja encontrar: "))
ultimo=1
penultimo=1

if (n==1) or (n==2):
    print("1")
else:
    for count in range(2,n):
        termo = ultimo + penultimo
        penultimo = ultimo
        ultimo = termo
        count += 1
    print(termo)
```

PS: algumas fontes definem como 0 o primeiro termo, 1 o segundo termo e do terceiro em diante é a soma dos dois termos anteriores, que é uma definição diferente dos dados da questão. No fim das contas, isso não importa muito, pois a lógica é a mesma, só precisa de alguns pequenos ajustes.

17.

O fatorial é uma operação matemática, representada pelo símbolo de exclamação: !

A fórmula de fatorial de um número n é:

$$n! = n * (n-1) * (n-2) * \dots * 3 * 2 * 1$$

Ou seja, o valor de $n!$ (leia-se: n fatorial) é o produto de 1, por 2, por 3, ..., por $(n-1)$ e n .

Por exemplo:

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120 \text{ (ou: } 5! = 5 \times 4! = 5 \times 24 = 120)$$

E por aí vai.

O fatorial é muito usado em um ramo da Matemática chamado Análise Combinatória. Muito mesmo. Se você fez ensino médio, certamente sabe o que é um fatorial.

Agora, vamos levar isso pro nosso amado Python.

Fatorial com o laço **WHILE** em Python

Vamos pedir um número inteiro e positivo ao usuário, e armazenar na variável *numero*.

Agora precisamos calcular o produto:

$1*2*3*...*(numero-1) * numero$

Vamos armazenar esse produto na variável *resultado*.

Inicialmente, vamos iniciar a variável resultado com 1:

resultado = 1

Agora vamos precisar de um contador, vamos usar a variável **count** para ir de 1 até *numero*, a cada iteração do laço WHILE, ela é incrementada.

Então, dentro do laço while, fazemos com que *resultado* seja multiplicado por *count*, pois *count* vai assumir os valores de 1, 2, 3, ..., até *numero*.

Quando chegar em *numero*, nosso looping while deve parar.

Logo, o teste condicional do while deve ser: ***count* <= *numero*** , ok?

Veja como ficou nosso script em Python:

```
numero = int(input("Fatorial de: "))
```

```
resultado=1
```

```
count=1
```

```
while count <= numero:
```

```
    resultado *= count
```

```
    count += 1
```

```
print(resultado)
```

Lembrando que:

• **resultado *= count** é o mesmo que: **resultado = resultado * count**

Ou seja, o novo valor de 'resultado' vai ser o valor antigo multiplicado por 'count'.

•Já: **count += 1** significa: **count = count + 1**

Ou seja, estamos incrementando o valor de 'count' em uma unidade, a cada iteração do laço WHILE.

Calcular fatorial usando laço **FOR** em Python

A lógica deve ser a mesma. Na verdade, tudo que um laço WHILE faz o FOR faz, e vice-versa.

"Ué, então pra que criar os dois?"

Porque tem situações que é melhor usar um e situações que é melhor usar outro.

Neste caso, você vai ver que é melhor usar o FOR.

Vamos repetir a ideia: Precisamos fazer várias multiplicações, do número 1 até o valor inserido pelo usuário, *numero*.
Do intervalo 1 até *numero*.

Isso te lembra algo? Intervalo?

Sim, **range**.

Nossa querida, amada **função range do Python**, que é unha e carne com o **laço FOR**.

Como queremos um intervalo que vai de 1 até *numero*, basta usarmos:
range(1, numero+1)

Veja como nosso código fica bem mais enxuto e fodástico usando o laço FOR:

```
numero = int(input("Fatorial de: "))
```

```
resultado=1
```

```
for n in range(1,numero+1):  
    resultado *= n
```

```
print(resultado)
```

21.

Um número é dito primo quando é possível dividir ele (divisão de inteiro com inteiro) por 1 e por ele mesmo.

Exemplos de números primos:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373 ...

Pode sair tentando dividir esses daí por outro número menor, que não seja 1 ou ele mesmo, que não vai conseguir.

Exemplos de números que não são primos:

4: É possível dividir por 1, 2, e 4

6: É possível dividir por 1, 2, 3 e 6

8: É possível dividir por 1, 2, 4 e 8

2112: É divisível por 2 3 4 6 8 11 12 16 22 24 32 33 44 48 64 66 88 96 132 176 192 264 352 528 704 e 1056

Primos: Para que servem ?

Não, não é um conjunto inútil e sem sentido de números.

Se fosse, nem seriam estudados.

O assunto número primo é um dos mais pesquisados, estudados e misteriosos da história da humanidade.

Até hoje, não se tem uma fórmula para se criar números primos.

Ainda não descobriram um 'padrão' definitivo.

Um dos usos mais importante é no ramo da criptografia, principalmente com o algoritmo do sistema RSA.

Existem dois tipos de cigarras que possuem ciclos de vida de 13 e 17 anos,

assim somente a cada 221 anos elas tem que dividir a floresta quando saem da terra, evitando se encontrar, o que prejudicaria sua permanência na natureza.

Enfim, se pesquisar na internet, vai achar uma infinidade de coisas onde os números primos estão metidos no dia-a-dia.

2	3	5	7	11
13	17	19	23	29
31	37	41	43	47
53	59	61	67	71
73	79	83	89	97

Inicialmente, pedimos um número inteiro e positivo para o usuário e armazenamos na variável **n**.

Vamos armazenar na variável **mult** o número de múltiplos que existe de 2 até **n-1**.

Ou seja, do intervalo (2, 3, 4, ..., n-1)

Isso é obtido usando a **função range**: **range(2,n)**.

Vamos usar a variável **count** pra receber cada um desses valores, dentro desse intervalo.

Dentro do looping, temos que testar se o resto da divisão de **n** por **count** vai ser 0. Se for, é porque **n** é múltiplo de **count**, logo não é primo.

A medida que nosso programa ai encontrando múltiplos, conta eles na variável **mult** e exibe na tela.

Após terminar o laço, testamos o valor de **mult**.

Se permanecer zerado, é porque o número fornecido pelo usuário é primo. Vejamos nosso código Python:

```
n = int(input("Verificar numeros primos ate: "))
mult=0

for count in range(2,n):
    if (n % count == 0):
        print("Múltiplo de",count)
        mult += 1

if(mult==0):
    print("É primo")
else:
    print("Tem",mult," múltiplos acima de 2 e abaixo de",n)
```

Função

O que é? Para que serve? Onde são usadas as funções ?

Neste [tutorial de Python](#), vamos iniciar nossos estudos em funções, que é de longe, um dos assuntos mais importantes em programação.

Inicialmente, vamos tentar te explicar de uma maneira bem simples e fácil o que é uma função em Python, para que serve, onde e como iremos usar elas durante nosso curso.

Vamos lá!

- **Função em Python - O que é ?**

Função nada mais é que um pedaço de código, um trecho com várias declarações e linhas de código, como já fizemos vááárias vezes aqui em nosso curso.

Porém, são trechos de códigos que tem um funcionamento diferente, eles são mais específicos e seu objetivo é desempenhar uma tarefa específica.

Não sei se estão percebendo, mas nossos código estão ficando cada vez mais longos, complexos, cheio de variáveis, testes, laços e tudo mais.

Um simples programa ou jogo, contém milhares de linhas de código.
Um sistema operacional, como Windows ou Linux, tem milhões. Isso mesmo, milhões de linha de código.

Imagina um arquivo com milhões de linhas de código? Imaginou?
Imaginação fértil você tem, então. Pois isso não existe.

la ficar muito grande, muito bagunçado, difícil de achar algo, se organizar.
E é aí que entra a função, para organizar isso, facilitar, deixar tudo mais fácil de se controlar.

Vamos te dar um exemplo real.

- **Empresa Python sem Função**

Como você estudou pelo **curso Python Progressivo**, obteve seu **certificado do curso** e começou a trabalhar como programador, rapidamente ficou rico.

Então decidiu contratar 30 funcionários para sua empresa. Todo mundo igual e todo mundo vai fazer o mesmo.

Quem chegar primeiro na empresa, abre ela, quem vem chegando depois começa a limpar, varrer, outro faz o café...e assim vai, todo mundo fazendo tudo.

Você começa a fechar contrato com alguma empresa, mas tem que parar tudo pra ir fazer mais café (programador adora café). E perde o contrato que tava fechando, acontece.

Seu melhor programador está ocupado, teve que ir no banco pagar uns boletos da empresa.

Sua melhor designer estava fazendo um trabalho fantástico...fazendo entrevistas de novos funcionários.

Ou seja, todo mundo faz tudo, o que aparecer, sem regra, é todo mundo igual, todo mundo faz tudo, todo mundo para uma coisa, ajuda o outro, começa isso, dá uma pausa, faz aquilo...

Você acha que isso vai dar certo? Uma empresa com pessoas sem **função** nenhuma? Experimenta ai e me diz se deu certo.

- **Empresa Python com Função**

Agora, após falir, voltar a ser empregado, você juntou dinheiro e decidiu tentar abrir uma empresa novamente. Mas agora, vai usar uma abordagem diferente, pois você estudou **funções em Python**.

Decidiu implantar um sistema com funções também, na sua empresa.

Primeiro, pegou um grupo de 5 pessoas. Colocou dois seguranças para se revezarem, e mais 3 para ficar limpando a empresa, fazendo café etc.

Pegou mais 2 pessoas para cuidar da tesouraria, eles recebem pagamento dos seus clientes, fazem transferência, vão no banco resolver assuntos financeiros, fazem pagamento dos clientes etc.

Definiu bem também a equipe de programadores, escolheu os melhores (que, obviamente, estudaram pelo Python Progressivo) e deixou eles focados apenas nisso.

O mesmo para sua equipe de design, mídia, áudio, vídeo etc.

Tem também a galera do RH, que contrata, faz entrevistas, cuida dos estagiários. E tem a parte burocrática, com advogados e contadores para abrir a empresa, calcular impostos, cuidar da parte trabalhista.

E aí, e agora, será que vai dar certo?

Com pessoas e grupos desempenhando **funções** específicas?

Eeeeeu acho que tem mais chances de funcionar assim! E você ?

- **Funções em Python - Para que servem ?**

Simplificação do código

As funções deixam seu projeto mais simples e fáceis de se manter e entender. Tem que programar algo na sua empresa? Chama a equipe cuja função é programar.

Assunto financeiro? Chama a função **tesouraria**, eles resolvem tudo isso lá. Quer contratar? A função **RH** vai entrevistar, depois passar pra **tesouraria** pagar o salário, depois a equipe de função **administração** vai definir onde essa pessoa vai trabalhar melhor.

Reuso de código

Quando você vai criar um novo projeto, você não precisa programar tudo do zero, tudo de novo. Sempre que um novo programa é criado em Python, boa parte do projeto já está pronto.

Por exemplo, já te ensinamos como **calcular fatorial em Python**. O que você

deve fazer? Guardar esse código específico, como uma **função**, cuja função é calcular o fatorial de um número fornecido.

Se em um novo projeto que vai começar hoje, mês que vem ou daqui 10 anos você precisar calcular fatorial novamente, não precisa programar tudo de novo, você já tem a função fatorial pronta. Só usar novamente.

Desenvolvimento em blocos e equipe

Vamos supor que você e seus amigos decidiram criar um jogo. Vocês podem, e devem, desenvolver tudo de maneira mais segmentada.

Um de você vai programar a parte de design, dos *skins*, das imagens dos personagens se mexendo. Cria várias funções, para cada coisa específica.

Outro, melhor em exatas, fica com a parte de fazer cálculos (calcular poder, destruição, defesa, acumular dinheiro, *skills* etc). Sempre tem muita matemática em um jogo. Crie funções específicas, só sobre isso.

Já outro vai trabalhar no banco de dados pra armazenar os dados do jogo, que fase você está, os recursos que você coletou etc, e cria funções que armazenam essas funções. Quando abre o jogo, ele vai buscar esses dados e carrega, pra começar de onde parou.

Uma outra pessoa fica responsável só pela parte de **internet**, pra se conectar, adicionar pessoas como amigos, jogar online, criar um chat. Cria funções específicas só com esse propósito.

E você? Você é o chefe do projeto...vai pegar os blocos de funções dos personagens, com as funções de design, mídia, integrar tudo usando as funções de banco de dados, fazer tudo isso rodar remotamente, através dos códigos de funções de internet.

Entendeu? Cada um faz um trecho, uma parte, criando coisas bem específicas, códigos específicos, funções! E depois junta tudo.

Facilidade de testes e Encontrar erros

Fez a parte do jogo responsável pela internet? Testa só ela.
Só aquele bloco de código, que faz algo específico. Com funções bem limitadas e simples, fica fácil de testar.

Deu pau no som? Não precisa mexer no banco de dados, vai nas funções de som. A imagem travou? Não precisa mexer nas funções de internet, só ir direto no código de imagens.

Execução ilimitada de código

Você digita o código da função apenas uma vez e ele é executado quantos vezes você desejar. Basta "chamar" a função sempre que você quiser.

Sabe jogos de luta? Quando você dá um golpe e sai um som?
Então, tem uma função que detecta se você deu um soco, um chute ou soltou um poder. Dependendo do que fez, ele chama outra função que vai emitir o som.

Quantas vezes esse som vai ser emitido?

Ué, quantas vezes você der o golpe...pode dar 1..10...100...mil...cada vez que você aperta um botão, funções são executadas.

Mas aí que vem a beleza: esse código é escrito apenas uma vez.
E ele é executado quantas vezes você quiser.

Não precisa ficar repetindo o código várias e várias vezes nos programas.
Faz só uma vez e executa aquele trecho sempre que for necessário, simples assim.

- **Funções - Onde são usadas ?**

Qualquer código, script ou programa mais complexo em Python, pode e deve se utilizar de funções.

Como você verá nos códigos daqui em diante, em nosso **curso de Python**, iremos utilizar bastante funções.

Além de todas as vantagens que citamos acima, elas são muito importantes

para **receber** dados, **tratar** essas informações (fazer alguma coisa específica com essas informações) e **retornar** valores.

É como se funções fossem uma caixinha mágica.

Você fornece algumas informações, como as notas de uma turma.

Lá dentro acontece uma mágica (código que calcula coisas, com a média).

E retornam valores (média da turma).

Você pode, inclusive, usar funções que foram desenvolvidas por outras pessoas.

Essas funções geralmente são **documentadas**, ou seja, alguém diz para que ela serve, que tipo de dados você deve fornecer ao chamar essas funções e o tipo de dado que ela retorna.

Ou seja, você pode trabalhar numa coisa. Seu amigo em outra, você usar funções que ele criou, ele usar funções que você programou, sem ao menos um ter olhado pro código do outro!

Tudo isso apenas sabendo o que cada função faz, depois é só usar ela!

Agora, vamos deixar de papo teórico e partir pro nosso próximo tutorial, onde vamos começar a usar, de fato, as danadas das funções!

Como Declarar, Chamar e Usar funções

Neste tutorial de nossa **Apostila de Python online e grátis**, vamos aprender como declarar uma função, além de ver como fazer para chamar ela, sempre que quisermos.

• [Clique aqui para saber como Trabalhar e Ganhar dinheiro como Programador Python](#)

• Como Declarar uma Função: **def**

Para criar uma função em linguagem de programação Python, precisamos fazer uma declaração correta e precisa dela.

O escopo da declaração de uma função é:

```
def nome-funcao(argumentos):  
    código  
    código  
    ...  
    return valor
```

A primeira linha é chamada de *header* ou cabeçalho. A definição da declaração da função começa com a palavra-chave **def**. O que vem após o *def* é o nome da função.

No nosso caso, o nome da função é **nome-funcao**.

Em seguida, um par de parêntesis, que pode conter **argumentos** (informações para a função usar).

Depois, os dois pontos. E abaixo, o código da sua função.

Por fim, a instrução **return** com alguma informação, um valor, para retornar para quem chamou a função.

Calma, com exemplos, e aos poucos, você vai entender tudo isso bem melhor.

Há duas coisas opcionais:

Argumentos (informações que você passa para a função, ao chamar ela)
Retornar valor (informações que a função retorna para quem chamou ela)

Como Chamar uma Função

Como você chama alguém? Uma pessoa, um amigo?

Pelo nome, correto?

Em funções, é a mesma coisa. Basta escrever o nome dela, e ela roda.
Simples assim.

Se declaramos uma função assim:

```
def lalala():  
    código  
    código
```

...

O nome dela é **lalala**, não tem argumentos nem retorna nenhum valor.
Para chamar ela, fazemos com a seguinte linha de código:

lalala()

E pronto. Só isso. Vamos ver um exemplo de código real?]

- **Exemplo de código com Função**

**Crie um programa em Python que exibe a mensagem "Olá, mundo!"
Porém, essa mensagem deve rodar de dentro de uma função.**

Vamos criar uma função chamada **mensagem**. O que ela faz é simplesmente exibir uma mensagem, um print.

Agora, basta chamar:

mensagem()

Pronto, só isso, veja como fica o código:

```
def mensagem():  
    print("Olá, mundo")
```

mensagem()

- **Script de Python com Funções**

Em todos nossos scripts e programas de Python, aconteceu uma coisa:
Ele rodava da primeira até a última linha.
Linha por linha.
Na ordem.

No exemplo de código anterior, não.
É definido uma função, tem um print...mas nada ocorre.

Só depois que chamamos a função (**mensagem()**), é que acontece algo.
Ou seja, o Python **pula** esses **def**, não roda ele de cara.

Aliás, se você não tivesse chamado a função via **mensagem()**, aquele código nem seria executado.

Quando chamamos uma função de nome **nome()**, o Python sai buscando no código uma **def nome()**, ou seja, ele sabe que você chamou uma função.

Então ele para tudo e vai buscar onde essa função foi declarada e executa ela.
Ok?

Pode ter 1 milhão de linhas de código com funções. Se elas não forem chamadas, simplesmente não executam.

Mas se tiver uma função de uma linha (como a do exemplo), e ela for chamada 1 milhão de vezes, ela é executada 1 milhão de vezes.

Ou infinita vezes. O código a seguir executa o ***Olá, mundo!*** infinitamente:

```
def mensagem():  
    print("Olá, mundo")
```

```
while 1:  
    mensagem()
```

- **Exemplo de Código com Funções**

Crie uma função que pede dois números, faz a soma e exibe o resultado, através de uma função. O usuário pode executar a função quantas vezes desejar.

A função que soma será a **soma()**. Ela é bem simples, pede **x**, pede **y**, transforma tudo em float, soma na variável **soma** e exibe este valor. Pronto. Função é isso, um código simples, pequeno e bem específico, nada mais, nada menos.

Agora vamos ao programa normal.

Talvez seja preciso chamar a função **soma()** várias vezes. Quem vai controlar isso é a variável **continuar**.

Inicialmente ela é 1 (verdadeiro ou TRUE).

Com um laço **while**, testamos o valor de **continuar**, como é 1, é TRUE, então o looping roda.

Dentro do laço, é testado o valor de **continuar**, se for verdadeiro, cai num **IF** que chama a função **soma()**.

Inicialmente, é tudo verdadeiro, logo a função é invocada.

Ela roda normalmente e faz tudo bonitinho.

Depois, perguntamos ao usuário se ele deseja continuar a fazer mais somas. Se desejar encerrar o programa, basta digitar 0.

continuar vai ser 0, o laço **while** vai ter teste FALSO e não vai mais executar, encerrando o looping e o programa.

Se digitar qualquer coisa diferente de 0, o teste do laço **while** vai ser verdadeiro e vai rodar de novo.

Nosso código fica:

```
def soma():  
    x = float(input("Primeiro numero: "))  
    y = float(input("Segundo numero: "))  
    soma = x + y  
    print("Soma: ",soma)
```

```
continuar=1
```

```
while continuar:  
    if(continuar):  
        soma()  
        continuar=int(input("Digite 0 se desejar encerrar ou qualquer outro numero  
para continuar: "))
```

Bacana, né?

- **Exemplo de Código com Funções**

Faça uma calculadora, usando funções. O script pergunta qual operação o usuário deseja rodar (soma, subtração, multiplicação ou divisão) e executa a operação.

A calculadora deve ser executada quantas vezes o usuário desejar.

A ideia aqui é criar várias funções, e dependendo do que o usuário digitar, executar a função corretamente.

Veja a solução completa:

Várias Funções em um Programa Python

A solução deste exercício é criar várias funções diferentes, uma para cada operação matemática.

Fizemos as funções **soma()**, **subtracao()**, **multiplicacao()** e a **divisao()** . O código delas, é bem simples e fácil de você entender, nessa altura do campeonato de nosso curso Python Progressivo.

A opção do usuário vai ser armazenada na variável **opcao**, que inicialmente é 1, para o primeiro teste no laço **while** ser verdadeiro.

Em seguida, o usuário deve digitar a opção que deseja realizar. Para identificarmos a opção digitada, usamos vários testes **IF**.

Aquele que for verdadeiro, chama a função corretamente. Se o usuário digitar 0, o programa acaba e o script se encerra.

Veja como ficou nosso código Python de calculadora usando funções:

```
def soma():  
    x = float(input("Primeiro numero: "))  
    y = float(input("Segundo numero: "))  
    print("Soma: ",x+y)
```

```
def subtracao():  
    x = float(input("Primeiro numero: "))  
    y = float(input("Segundo numero: "))  
    print("Subtracao: ",x-y)
```

```
def multiplicacao():  
    x = float(input("Primeiro numero: "))  
    y = float(input("Segundo numero: "))  
    print("Multiplicacao: ",x*y)
```

```
def divisao():  
    x = float(input("Primeiro numero: "))  
    y = float(input("Segundo numero: "))  
    print("Divisao: ",x/y)
```

```
opcao=1
```

```
while opcao:
    print("0. Sair")
    print("1. Somar")
    print("2. Subtrair")
    print("3. Multiplicação")
    print("4. Divisão ")

    opcao = int(input("Opção: "))

    if(opcao==1):
        soma()
    if(opcao==2):
        subtracao()
    if(opcao==3):
        multiplicacao()
    if(opcao==4):
        divisao()
```

Função chamando Função

Neste [tutorial de Python](#), vamos mostrar mais uma técnica interessante e bastante importante: chamar uma função, dentro de outra função.

- **Função no Script: O que acontece ?**

Até antes desta seção sobre funções, nossos scripts rodavam da seguinte maneira: o interpretador Python ia lendo e executando, da primeira para última instrução, linha por linha.

Agora, com funções dentro do script, coisas diferentes acontecem por 'trás' dos panos. Vamos voltar a ver um exemplo simples de código:

```
def mensagem():  
    print("Tutorial Python Progressivo")
```

```
mensagem()
```

O que acontece agora, é o seguinte:

O interpretador continua lendo da primeira para última linha, porém, quando encontra uma função (**def**), ele não executa.

Ele percebe que é uma função, e aloca na memória do seu computador um espaço para ela, e toma 'ciência' que ela existe:

"Opa, esse script tem uma função...xô guardar o nome dela e seu código aqui nesse endereço do computador. Quando chamarem, já sei onde ela tá"

Em seguida, ele continua a interpretar o resto do script.

Quando chega na instrução **mensagem()**, ele executa normalmente.

No caso, vai lá no local onde armazenou a função, e agora sim executa o código da função.

No exemplo de nossa [calculadora em Python usando funções](#), declaramos 4 funções: **soma()**, **subtracao()**, **multiplicacao()** e **divisao()**.

Cada vez que o interpretador se deparava com a declaração de uma função, armazenava ela na memória, mas sem executar.

Somente depois de ler todas as declarações, é que começou a executar as instruções, que vieram depois das funções. Veja lá o artigo anterior.

É importante saber como o interpretador funciona, para ser um bom programador Python. Ok ?

- **Função chamando Função**

Quando explicamos **o que é uma função em Python**, demos o exemplo da empresa, que cada setor e grupo de pessoas são uma função, que faz algo específico.

Vamos supor que chegou um candidato para trabalhar na empresa. Na entrada, ele chega na secretaria da empresa (uma função), se identifica. Ela vai redirecionar a pessoa para a entrevista com o RH (outra função), aprovando o candidato, o mesmo vai ser encaminhado para algum programador fazer a entrevista sobre os conhecimentos técnicos dele, esse programador também é uma função específica.

Ou seja: funções 'conversam' entre si, uma chama a outra, elas podem e devem trabalhar juntas!

No exemplo de código a seguir, vamos declarar duas funções: **mensagem()** e **python_progressivo()**, onde ambas exibem mensagens na tela:

```
def mensagem():  
    print("O melhor curso de todos é:")  
    python_progressivo()  
    print("Estude por lá!")
```

```
def python_progressivo():  
    print("Curso Python Progressivo")
```

```
mensagem()
```

Vamos lá, ver passo a passo o que ocorre.

Inicialmente, o interpretador vai ler a função **mensagem()**, anota seu nome e guarda seu código num endereço de memória.

Depois faz o mesmo com a função **python_progressivo()**.

Como são declarações de funções, não são executadas. Funções só são executadas quando são chamadas, ok?

E, por fim, ele vai executar uma instrução, a linha de comando: **mensagem()**, responsável por invocar a função **mensagem()**.

Agora o interpretador vai lá no código dessa função pra executar tudo que tem dentro. Primeiro, manda um print na tela "O melhor curso de todos é: "

Na instrução seguinte, ele chama a função **python_progresso()**. Agora o interpretador vai nos levar lá pro código dessa função e executar ela. No caso, exibe a mensagem "Curso Python Progressivo" e encerra essa função.

Quando encerra a execução da **python_progressivo()**, o interpretador volta pra onde tava, antes de chamar ela, no caso, para dentro da **mensagem()**, cuja próxima instrução é printar na tela "Estude por lá".

Agora a função **mensagem()** se encerra e volta lá pra baixo, onde ela foi chamada. Como não tem nada depois, o script Python se encerra.

Variável Local

- **Variável em Funções**

Em praticamente todos os scripts que fizemos, usamos variáveis. Seja pra trabalhar com números inteiros, quebrados, strings etc, mas sempre usamos.

Usávamos em uma linha de código, depois alterávamos o valor em outra linha e por aí ia, sem mistério.

Mas quando começamos a trabalhar com funções, algo de diferente acontece.

Vamos ver um exemplo.

Vamos armazenar um valor na variável **var** dentro da função **teste()**. Colocamos o número inteiro 2112 dentro dessa variável e damos um print nesse valor, ainda dentro da função.

Fora da função, vamos simplesmente dar um print nessa variável, veja o código:

```
def teste():  
    print("Estamos dentro da função!")  
    var=2112  
    print("Valor de var:",var)  
  
teste()  
print("Agora estamos fora da função!")  
print("Valor de var:",var)
```

Agora veja o resultado:

```
Estamos dentro da função!  
Valor de var: 2112  
Agora estamos fora da função!  
Traceback (most recent call last):  
  File "/home/user/funcao.py", line 8, in <module>  
    print("Valor de var:",var)  
NameError: name 'var' is not defined  
>>> |
```

Dentro da função...tudo ok! Definimos a variável, atribuímos um valor a ela, imprimimos e tudo certo.

Fora da função, não! Deu ruim!!! E agora?

- **Variável Local em Python**

A explicação é muito simples: as variáveis dentro de uma função, não são visíveis fora da função.

Se você criou e usou uma variável dentro de qualquer função, ela só existe naquele escopo, ou seja, apenas ali dentro da função.

Quem está de 'fora', não sabe que ela existe, não sabe o que tem dentro da função.

Vamos fazer mais um teste, com 3 funções. Uma função chama a outra. E em cada função vamos usar uma variável de mesmo nome, a **banda**, veja:

```
def teste():  
    banda="Rush"  
    print("A melhor banda do mundo é ", banda)  
    teste2()
```

```
def teste2():  
    banda="Beatles"  
    print("A melhor banda do mundo é ", banda)  
    teste3()
```

```
def teste3():  
    banda="Iron Maiden"  
    print("A melhor banda do mundo é ", banda)
```

```
teste()
```

Ou seja, cada função é um 'universo'. Outras funções ou outro escopo, não sabem que se passam lá dentro, **nestes casos**.

Existem, porém, uma maneira de funções se comunicarem com outras. Mas isso é assunto para o próximo episódio (sempre quis dizer isso).

Argumentos e Parâmetros nas Função

Neste [tutorial de Python](#), vamos te explicar o que são **argumentos** e **parâmetros**, para que servem, como funcionam e como usar, um dos assuntos mais importantes desta seção de funções em Python.

Veja também: [Como se tornar programador profissional em Python](#)

- **Argumentos e Parâmetros em Funções Python**

No tutorial anterior, falamos das variáveis locais, no caso, dentro de funções. Vimos que quando declaramos e usamos uma variável dentro de uma função, ela não é 'vista' do lado de fora.

Se tentar imprimir ou usar uma variável que está dentro do escopo de uma função, não vai conseguir. Nem o contrário, se estiver dentro de uma função, não vai conseguir usar variáveis e dados que estão fora da função.

Porém, isso contradiz o que falamos de funções, no início de nossos estudos sobre esse assunto, pois lá explicamos que elas se comunicam entre si, recebem dados, fazem tarefas específicas e retornam dados, ou seja, se comunicam.

Então, vamos aprender agora como fazer para funções se comunicarem com o mundo externo. Uma das maneiras de comunicação é através do uso de **argumentos** e **parâmetros**.

- **Argumento**

Argumento nada mais é que uma informação que passamos para uma função, quando chamamos ela. Enviamos dados (objetos) quando invocamos a função, e ela vai poder usar essas informações do jeito que quiser.

- **Parâmetro**

Quando você vai definir uma função e deseja que ela receba dados, você precisa definir **parâmetros** nessa função, um ou mais parâmetros. Uma

forma de dizer que aquela função só funciona se receber determinadas informações (parâmetros).

- **Argumento e Parâmetro - Como Usar**

Lembra quando ensinamos a declarar uma função?

Usamos a palavra-chave **def**, depois damos o nome da função e parêntesis? E dentro desses parêntesis não tinha nada?

Pronto, agora a diferença vai vir aí.

Dentro desses parêntesis é que a magia vai ocorrer.

A nova definição é:

```
def nome_da_funcao(parametro1, parametro2, parametro3,...)
    código da função
    código da função
    código da função
```

Você pode usar quantos parâmetros quiser: um, dois, três...basta especificar na definição da função, o número de parâmetros que ela vai receber.

E para invocar, chamar essa função, você deve fazer:

```
nome_da_funcao(argumento1, argumento2, argumento3,...)
```

Ou seja, se a função foi declarada como tendo 'x' parâmetros, devemos passar 'x' informações, ou seja, 'x' argumentos ao invocar a função.

Exemplo de Função com Parâmetro

A função seguinte recebe um parâmetro, vamos chamar ele de **x**, em seguida mostra o dobro desse valor:

```
def dobro(x):
    dobro = 2*x
    print("Dobro =",dobro)
```

Para invocar essa função, precisamos fornecer um valor, por exemplo:
dobro(1)

Iremos ver a mensagem na tela: **Dobro = 2**

Neste caso, **x** é um parâmetro, pois está definido na função.
Já o valor **1** que passamos, é o argumento.

- **Vários Argumentos e Parâmetros**

Você pode passar para uma função quantas informações desejar.
Pode passar 1 argumento, dois, dez, mil, 1 milhão... o que quiser.

Só é necessário fazer uma coisa: ao definir a função, deixar bem claro no cabeçalho dela quantos argumentos ela vai receber, colocando o número correto de parâmetros, bem como seus nomes e separados por vírgulas.

Por exemplo, a função a seguir vai receber dois argumentos, pois foi definida com dois parâmetros, **x** e **y**:

```
def soma(x,y):  
    soma = x + y  
    print("Soma =",soma)
```

Ela recebe dois argumentos e exibe a soma deles.

Na hora de invocar a função, devemos passar duas informações, ou seja, dois valores, dois argumentos, pois foi assim que a função foi definida, para receber dois dados! Por exemplo:

- soma(1,2)
- soma(-1,1)
- soma(21,12)

Se ela foi definida com **n** número de parâmetros, ela deve receber **n** número de argumentos, mais ou menos que isso, dará erro na sua função. Ok?

- **Exemplo de Função com Parâmetro e Argumento**

*Crie um programa que peça ao usuário dois valores. Estes números deverão ser repassados para uma função chamada **calculadora**, que vai mostrar a soma, subtração, divisão e multiplicação desses números.*

Nossa função deve ter dois parâmetros. Vamos chamar eles de **x** e **y**.

Fora da função, vamos pedir dois números ao usuário e armazenar nas variáveis **primeiro** e **segundo**.

Em seguida, passamos essas variáveis como argumentos, para a função **calculadora**. Veja como ficou nosso código:

```
def calculadora(x,y):  
    print("Soma      =",x+y)  
    print("Subtração  =",x-y)  
    print("Divisão    =",x/y)  
    print("Multiplicação =",x*y)  
  
primeiro = float(input("Primeiro número: "))  
segundo  = float(input("Segundo número: "))  
  
calculadora(primeiro,segundo)
```

- **Nomes de Parâmetros e Argumentos**

Note uma coisa extremamente importante no exemplo anterior:
Passamos as variáveis **primeiro** e **segundo** como argumentos.

Porém, ao chegar na função **calculadora**, ela vai tratar esses valores com outros nomes de variáveis: **x** e **y**.

Na definição da função, os nomes dos parâmetros são **x** e **y**, então dentro da função, serão tratados apenas como **x** e **y**, embora tenha passado (1,2) ou (primeiro,segundo) ou (jose,joao) como argumento.

O nome dos argumentos não importa, vão virar o nome dos parâmetros, que você deixou bem claro ao definir sua função.

Exercício Resolvido de Funções com Parâmetros e Argumentos

Crie um script em Python que pede uma frase (string) ao usuário e em seguida um caractere. Em seguida, seu script deve dizer quantas vezes aquele caractere apareceu na frase digitada. Use função com parâmetros.

Primeiro, vamos definir nossa função, a **conta_caractere**.

Ele deve ter dois parâmetros o **texto**, que é uma string e **char** que é um caractere.

Vamos usar a variável interna **count** para contar quantas vezes **char** aparece na string **texto**, usamos um laço while para fazer isso, e cada vez que encontra, o contador é incrementado em uma unidade.

Usamos um laço **for** de uma maneira bem interessante, vejam:

```
def conta_caractere(texto,char):  
    count = 0  
    for letra in texto:  
        if letra == char:  
            count += 1  
    print(char,"apareceu",count,"vezes na string")
```

```
string=input("Digite um texto qualquer: ")  
caractere = input("Digite um caractere: ")
```

```
conta_caractere(string,caractere)
```

Os dados solicitados ao usuários foram armazenados nas variáveis **string** e **caractere**, que foram passadas como argumentos para a função **conta_caractere**. Simples, não?

- **Exercícios Propostos**

01. Crie uma função que recebe um número e exiba seu quadrado.
02. Crie uma função que recebe um valor e exiba seu cubo.
03. Crie uma função que recebe 4 notas de um aluno, e exiba a média dele.
04. Crie uma função que recebe 3 números e exiba o maior deles.

Passagem de Argumentos por Valor em Funções

Nesta aula de nosso **Curso de Python**, vamos aprender um pouco mais sobre argumentos, parâmetros e funções, especificamente sobre um troço chamado *passagem por valor*.

•Leia também: [Certificado do Curso de Python](#)

•Valores de Argumentos e Parâmetros

No tutorial passado, aprendemos o que são parâmetros e argumentos, e como usar eles em nossas tão amadas e queridas funções em Python.

Vimos que para podermos passarmos dados para funções, estas precisam ser declaradas com *parâmetros* apropriados, aí enviamos as informações como *argumentos*.

Mas vamos te mostrar um teste bem interessante, para te explicar melhor o que é passagem por valor.

No exemplo a seguir, no escopo geral, declaramos uma variável **var** com o valor de 1. Em seguida, passamos ela pra função **muda**, que vai mudar o valor da variável de 1 para 0.

Após a função terminar, voltamos para o escopo geral e imprimimos a variável novamente, para saber que valor ela tem. Veja o código:

```
def muda(var):  
    var=0  
    print("Valor de x dentro da função: ",var)  
  
var=1  
print("Valor de x antes da função: ",var)  
muda(var)  
print("Valor de x ao sair da função: ",var)
```

O resultado é o seguinte:

```
Valor de x antes da função: 1
Valor de x dentro da função: 0
Valor de x ao sair da função: 1
>>> |
```

Olha que curioso!

O valor de **var** começa com 1. Mandamos essa variável via argumento para a função, lá dentro ela muda para 0.

Quando essa função termina, imprimimos novamente **var**, ele voltou a ser 0!

Bugou o Python ?!

Calma, vamos entender melhor o que está acontecendo.

- **Passagem por Valor em Python**

O que ocorre é o seguinte:

Quando enviamos um argumento para uma função, ela vai ficar apenas com o **valor** dessa variável.

Ela vai copiar esse valor para outra variável.

Embora passemos a variável **var** para a função, e dentro da função ela use uma variável de nome **var**, são duas variáveis totalmente diferentes, armazenadas em valores diferentes na memória.

O Python pensa assim:

"Olá, sou uma função marota.

Opa, fui invocada, me chamaram, vou lá trabalhar.

Me enviaram aqui um argumento, uma tal de **var**, vou copiar o valor que tá armazenado nela (1) para minha própria variável **var**. Pronto, copiado.

To nem aí pra essa outra variável **var** aí, fora do meu escopo. Pra mim, ela nem existe, só peguei a cópia dela, seu valor."

Ou seja, quando passamos informações, passamos, na verdade, somente o **valor** da variável. Por isso se chama passagem de argumento por valor.

Existe outra maneira de passar informações, chamada passagem por

referência. Aí sim, passamos de fato a variável, o endereço dela da memória, então quando 'mexemos' nela dentro da função, o valor da variável original é realmente alterado.

Vamos aprender isso somente depois, por hora é importante você entender o que acontece quando enviamos um argumento para uma função e o que ela faz, de fato, por trás dos panos.

Aqui, no Curso Python Progressivo, é assim: você realmente aprende a linguagem, como ela funciona, o que as coisas realmente fazem.

Nada de decorar sem sentido, aqui a gente entende as coisas.

Argumento Posicional e Argumento Nomeado em Funções

- Argumento Posicional em Python

Nos tutoriais e exemplos usados até o momento em nossa apostila, fizemos uma relação entre parâmetros e argumentos.

Por exemplos, se definimos a função:

```
def fu_bar(par1, par2, par3)
```

E depois chamamos ela passando os argumentos:

```
fu_bar(4,5,6)
```

Quer dizer que, na função **fu_bar**, os parâmetros receberão os seguintes valores:

- par1 = 4
- par2 = 5
- par3 = 6

Ou seja, na ordem. O primeiro parâmetro recebe o primeiro argumento, o segundo parâmetro recebe o valor do segundo argumento, e assim sucessivamente.

Isso é chamado de **argumento posicional**, por motivos óbvios. E existe somente essa maneira de relacionar os valores de parâmetros com os valores enviados como argumentos.

Porém, o Python é uma linguagem absurdamente flexível e poderosa, então, existem mais maneiras de um parâmetro receber um valor de argumento, quando se usa vários.

- Argumentos Nomeados em Python

Para ilustrar melhor, vamos criar uma função que recebe 3 notas diferentes: **mat**, **fis** e **qui**, notas de um aluno em Matemática, Física e Química e exibe

sua média:

```
def media(mat,fis,qui):  
    media = (mat+fis+qui)/3  
    print("Média: ", media)
```

Ao chamar essa função, podemos passar os seguintes argumentos:

media(8, 9, 10)

Sabemos que as notas serão:

- mat = 8
- fis = 9
- qui = 10

Porém, existe uma outra maneira de passar os argumentos, em outra ordem, basta **nomear** o argumento com o nome do parâmetro que desejar, veja:

•**media(qui=10, mat=8, fis=9)**

Prontinho. O resultado é exatamente o mesmo.

As maneiras seguintes também exibem o mesmo resultado:

•**media(qui=10, fis=9, mat=8)**

•**media(fis=9, mat=8, qui=10)**

Ou seja, **não importa a ordem** se você usar argumentos nomeados! Aliás, dá até pra misturar argumentos nomeados com argumentos posicionais.

• Regra para Argumentos Nomeados e Posicionais

O que acha que vai ocorrer se usar:

•**media(8, fis=9, qui=10) ?**

O que diz sua intenção? Lembre-se: Python é uma linguagem bem intuitiva. Sim: o 8 vai ser atribuído ao primeiro parâmetro, que na definição da função é **mat**, e vai ter o mesmo resultado.

Mas existe uma importante regra que deve ser obedecida:

- Argumentos posicionais devem vir antes

Se fizer:

media(8, fis=9, 10)

Vai dar erro. Todos os argumentos posicionais devem vir antes. No caso, tentamos usar no primeiro argumento e no terceiro, vai dar erro, não pode!

Porém você pode fazer:

media(8, 9, qui=10)

Os argumentos que não são nomeados, são automaticamente chaveados de maneira sequencial, posicional, com a ordem dos parâmetros. Bem intuitivo, não acha?

Variável Global

Nos artigos anteriores de nosso curso de Python, falamos bastante sobre variáveis, escopo, que as variáveis dentro de uma função podem existir somente lá, que ao fazer uma passagem por valor para uma função, o valor original da variável não é alterado etc etc.

Mas, por muitas vezes, é bem útil ter variáveis que 'existem' e são visíveis para todos os elementos de um programa em Python.

E é para isso que servem as variáveis globais.

Se uma variável é definida e usada somente dentro de uma função, ela é dita **variável local**.

Então, por lógica, para uma variável ser global e para poder ser usada em todas as funções, ela deve ser definida **fora** das funções.

- **Keyword **global****

Para usarmos uma variável global, devemos fazer, basicamente, duas coisas.

Definir a variável fora do escopo das funções

Dentro da função, usar a linha de comando:

global nome_variavel

Onde 'nome_variavel' é o nome de sua variável global.

Vamos ver um exemplo para entender melhor.

- **Exemplo de variável global**

Escreva um script em Python que pergunta o preço de um produto e mostre:

Preço original

Desconto em porcentagem

Valor do desconto

Preço com o desconto

O desconto quem define é o gerente, na forma de variável global.

Vamos lá.

Nossa variável global se chamará 'desc' e terá inicialmente o valor 10 (10%).
Dentro da função **desconto()**, precisamos inserir a linha:
global desc

Isso diz ao Python:

"Ei Pyzinho (você é íntimo já), essa variável 'desc' é global, tá?

Usa aquela que foi definida fora da função, pelo amor de deus, não vai usar uma variável local!"

Veja como fica nosso código:

```
# Desconto do gerente  
desc = 10 # 10%
```

```
def desconto(valor):  
    global desc  
    print("Preço original : R$ ",valor)  
    print("Desconto      : ", desc,"%")  
  
    desc /= 100  
    print("Valor desconto  : ", valor*desc)  
    print("Preço c/ desconto: ", valor*(1-desc) )  
  
val = float(input("Preço do produto: "))  
desconto(val)
```

- **Variável Global - Onde usar ? Para que servem ?**

Como o nome diz, ela serve para ser 'vista' e alterada por todos, por qualquer função.

Se você tiver um projeto onde uma informação deve ser vista por todos e eles possam alterar essa variável, use a **global**. Porém, cuidado.

De uma maneira geral, é complicado usar ela, pois todos tem acesso, todos podem mexer nela, e isso pode tornar o processo de *debugging* (achar erros e problemas) bem complicado, afinal, o problema pode vir de qualquer lado.

Constante Global

Em muitas linguagens, existe um recurso chamado **constante global**, que nada mais é que uma variável que é definida como um determinado valor. E pronto, esse valor não pode mais ser alterado.

Ele vai ser o mesmo, para qualquer parte de seu código e não é possível mexer nele.

No Python, não tem uma maneira simples, clara e específica de se fazer isso, mas podemos meio que 'simular' fazer isso com variáveis globais.

- **Como Usar Constante Global**

Para fazer uso de uma constante global, é bem simples.
Basta você definir a variável fora do escopo das funções.

Ela não deve ser definida em nenhuma função. Usada nas funções, sim, ok. Mas a primeira vez que ela aparece deve ser, de preferência, bem no início do seu script, antes das definições das funções.

Depois, basta simplesmente usar a variável, sem usar a keyword global.

- **Exemplo de uso de constante global**

Crie um programa em Python que peça o raio de uma circunferência ao usuário, em seguida exiba o perímetro e a área desse círculo.

Inicialmente, vamos definir uma variável global, uma constante, o valor de ***pi***, que é 3,14

Depois, criamos a função de perímetro ($2 \times \pi \times \text{raio}$) e a de área ($\pi \times \text{raio}^2$). Bem simples, veja:

```
pi = 3.14
```

```
def perimetro(raio):  
    print("Perimetro: ", 2*pi*raio)
```

```
def area(raio):  
    print("Área: ", pi*raio*raio)
```

```
raio = float(input("Raio do círculo: "))
```

```
perimetro(raio)  
area(raio)
```

Se o seu jogo tiver, por exemplo, um número fixo de jogadores, esse valor pode ser armazenado em uma variável constante e global, afinal, nunca vai se alterar. Assim, todos podem acessar essa variável.

Retornar um Valor em uma Função

No início de nossa seção de **Função em Python**, explicamos que uma função tem dois propósitos básicos:

1. Fazer tarefas específicas (de preferência, da maneira mais clara e concisa o possível)
2. Se comunicar com outras partes do programa

Já fizemos funções com as duas coisas. No caso da 'comunicação', aprendemos a usar **parâmetros e argumentos**, assim vimos que é possível enviarmos informações para dentro de uma função.

Ou seja, é como se a função agora pudesse ter contato com o que existe fora de seu escopo, pois pode receber informação.

Mas algo está faltando aí, veja se você descobre...

As funções devem se comunicar.

Já aprendemos como elas **recebem** informações.

Mas pra uma comunicação completa, além de **receber**, ela precisa **retornar** informação.

Ele recebe dados de fora da função, mas não vimos funções enviando dados para **fora** da função.

E é aí que entra a *keyword* **return**.

Ela serve para a função retornar algum valor!

Vamos ver como fazer isso.

- **Como usar *return* em uma Função no Python**

A declaração de uma função que tem o comando de **return** é:

```
def nome_funcao(parâmetros):  
    código  
    código  
    código  
    ...  
    return algo
```

A única coisa nova e diferente, é a última linha da função:

return algo

'algo' pode ser qualquer variável, número, expressão, objeto etc.
Esse 'algo' vai ser retornado para onde a função foi invocada.

Vamos ver um exercício resolvido e comentado para entender melhor.

- **Exemplo de uso de *return***

*Crie um programa em Python que tenha a função **soma(x,y)** que recebe dois números e retorna o valor da soma deles.*

Uma função que recebe dois números e calcula a soma, você já sabe fazer.
A única diferença é que vamos dar um **return** nesse valor da soma,.

E se estamos **retornando** algo, alguma coisa precisa receber esse retorno.
Quem vai 'receber' o 'return' será a variável **res**.

Veja como fica nosso código:

```
def soma(x,y):  
    result = x+y  
    return result  
  
a = int(input("Primeiro numero: "))  
b = int(input("Segundo numero : "))  
res = soma(a,b)  
print("Soma: ", res)
```

A função retorna a soma **x+y**, armazenada na variável **result**, ou seja, retorna um número.

Esse número, armazenamos na variável **res**, depois simplesmente imprimimos essa variável.

Se quisermos, podemos usar direto a função **soma(x,y)** dentro do print e encurtar ainda mais a função **soma()**:

```
def soma(x,y):  
    return (x+y)  
  
a = int(input("Primeiro numero: "))  
b = int(input("Segundo numero : "))
```



```
print("Soma: ", soma(a,b))
```

Usamos **return x+y** pois, como dissemos, podemos retornar uma expressão diretamente, sem necessitar criar uma variável dentro da função.

- **Exemplo de uso de `return` com Strings**

*Crie um programa em Python que peça o **nome** e o **sobrenome** de uma pessoa, depois exiba na tela a mensagem "Olá **sobrenome, nome**".*

Ou seja, ele inverte a ordem do nome e sobrenome.

Vamos criar a função **inverte**, que recebe duas strings.

Dentro dessa função, criamos uma nova string a **nomeInverso**, que é criada a partir do parâmetro **sobrenome**, uma vírgula e o parâmetro **nome**.

Retornamos essa nova string.

Já o código que chama essa função, vai receber o retorno dela e armazenar na variável **invertido**, em seguida imprimimos essa variável, veja como fica:

```
def inverte(nome, sobrenome):  
    nomeInverso = sobrenome+", "+nome  
    return nomeInverso
```

```
nome = input("Nome: ")  
sobrenome = input("Sobrenome: ")  
invertido = inverte(nome,sobrenome)
```

```
print("Olá", invertido)
```

- **Exemplos de `return` com Booleanos**

*Crie um programa em Python que diz se o número inserido pelo usuário é par ou ímpar. Ele deve fazer isso através de uma função que recebe o inteiro e retorna **True** ou **False**.*

Vamos chamar nossa função de **par(x)**, ela recebe um número como argumento.

Dentro dela, usamos um **teste condicional IF ELSE**:

Se o resto da divisão de **x** por 2 for 0, ele é par

Senão, é ímpar

Se for verdadeiro, damos um **return True**.

Se for falso, fazemos **return False**.

Veja que nessa função dois **return**, mas somente um será executado, dependendo se **x** é par ou ímpar.

Na parte principal de nosso script, vamos usar outro teste condicional IF ELSE. O teste de condição é simplesmente **par(num)**, onde **num** é um número que o usuário vai digitar.

Como essa função retorna sempre um valor Booleano (valor lógico), podemos usar a chamada de inversão diretamente no teste do IF.

Se o número for par, essa chamada vai retornar **True** e o IF será executado, printando uma mensagem dizendo que o número é par.

Se for falsa, o IF não roda então vai pro ELSE, que roda, printando uma string dizendo que é ímpar.

Nosso código ficou assim:

```
def par(x):  
    if (x%2)==0:  
        return True  
    else:  
        return False  
  
while True:  
    num = int(input("Insira um número: "))  
    if par(num):  
        print("É par")  
    else:  
        print("É ímpar")
```

- **Como retornar mais de um valor**

Diferente da maioria das outras linguagens, onde só é possível retornar um valor, ou seja, apenas uma coisa, seja lá o que for, no Python podemos retornar várias variáveis, objetos, expressões etc.

Para isso, basta separar cada informação por vírgula.

Por exemplo, o script abaixo tem uma função de cadastro.

Ela pergunta primeiro seu nome, depois sua idade.

Então, retorna esses dados:

return name, age

Quem chamou essa função, tem que estar preparado para receber essas duas informações:

nome, idade = cadastro()

A variável **nome** vai receber o primeiro valor retornado, que é **name**, e a variável **idade** vai receber o segundo valor retornado da função, que é **age**.

Veja como ficou nosso código:

```
def cadastro():  
    name = input("Qual seu nome: ")  
    age = int(input("Idade: ") )  
  
    return name, age  
  
print("Iniciando cadastro...")  
nome, idade = cadastro()  
  
print("Cadastro realizado com sucesso:")  
print("Seu nome é", nome, "e você tem", idade, "anos de idade.")
```

É importante notar o objetivo dessa função **cadastro()**.

Sempre que você quiser fazer um cadastro, basta chamar essa função, que ela vai perguntar as coisas e retornar as informações importantes.

Você só faz chamar, e depois já tem os dados nas variáveis **nome** e **idade**. Obviamente, é um exemplo simples, mas imagine numa grande empresa,

onde você foi encarregado de contratar um novo programador.

Alguns dados pessoais, como salário, não são da sua conta.

Então você simplesmente chama uma função lá da tesouraria e ela fica encarregada dessa parte financeira.

Se você tá programando um jogo e está responsável pela parte artística, não tem que se preocupar com a parte de lógica...tipo, a velocidade com que seu carro se movimenta, isso uma outra galera vai fazer, você só vai usar as funções deles:

- acelerar()**
- frear()**
- trocar_marcha()**

O que acontece lá dentro? Sei lá! O código lá dentro é de outra pessoa, você só precisa saber que parâmetros a função pede e o que ela vai retornar, para poder usar uma função.

Interessante esse **return**, não é mesmo?

Agora sim nossas funções estão interessantes!

Elas recebem dados, fazem tarefas específicas e retornam informações.

Quem chama uma função, não precisa saber o que se passa dentro dela, o importante é fornecer os dados necessários (argumentos) e saber que tipo de informação vai receber (**return**).

Recursividade

"Para aprender recursividade, você precisa saber recursividade..."

Essa frase pode soar bem louca, a primeira vista.

Mas quando te ensinarmos melhor o conceito de uma função recursiva, ela vai fazer total sentido para você.

- **Somatório na Matemática**

Definimos como somatório, uma função matemática representada por:

$$f(x) = 1 + 2 + 3 + \dots + (x-1) + x$$

Ou seja, somamos tudo de 1 até o valor x .

Exemplos:

$$f(4) = 1 + 2 + 3 + 4 = 10$$

$$f(5) = 1 + 2 + 3 + 4 + 5 = 15$$

$$f(6) = 1 + 2 + 3 + 4 + 5 + 6 = 21$$

etc

Porém, note uma coisa:

$$f(5) = f(4) + 5 = 10 + 5 = 15$$

$$f(6) = f(5) + 6 = 15 + 6 = 21$$

Ou seja, podemos generalizar:

$$f(x) = f(x-1) + x$$

Agora, calma.

Vamos ver o motivo dessa teoria toda.

- **Função Recursiva em Python**

Dizemos que uma função é recursiva quando, dentro dela, ela chama ela mesma.

Porém, com outro argumento.

No exemplo anterior, passamos um valor x para a função recursiva.

Ela soma x com o valor de $f(x-1)$, ou seja, ela vai chamar ela mesma, mas ao invés de passar o parâmetro x , vai passar o parâmetro $x-1$.

E essa função que recebeu **x-1** vai somar (**x-1**) com **f(x-2)**, ou seja, chamou ela mesma novamente, mas agora com um parâmetro (**x-2**). E por aí vai.

Onde termina isso?

Ora, com $f(1) = 1$

- **Somatório com Função Recursiva**

Se ainda está perdido, tudo bem, é pra estar mesmo.

Mas vamos resolver um exemplo que vai te fazer entender melhor.

Crie um script que peça um inteiro positivo para o usuário.

Em seguida, exiba a soma do somatório de 1 até esse número.

Nosso código fica assim:

```
def somatorio(x):  
    if x==1:  
        return 1  
    else:  
        return x + somatorio(x-1)  
  
while True:  
    x = int(input("Somatorio de 1 até: "))  
    print("Soma: ",somatorio(x) )
```

Agora vamos ver o que faz linha por linha.

Primeiro, definimos nossa função, vai se chamar **somatorio** e recebe um valor como parâmetro, o **x**.

A primeira coisa a se fazer numa função recursiva é definir onde ela vai parar, ou seja, dizer "ei, chega, aqui você vai parar de invocar você mesma"

Nesse caso, é quando o argumento for **1**, aí o somatório é 1 e retorna 1.

Se não for argumento 1, ai cai no ELSE, então o retorno é o próprio argumento **x** somado de **somatorio(x-1)**.

Lembra da função $f(x)$? Agora ela se chama é *somatorio*:

$somatorio(x) = x + somatorio(x-1)$

E prontinho, no cordo do script pedimos um número ao usuário.
Por exemplo, se $x = 4$:

Primeiro return: $4 + \text{somatorio}(3)$

Segundo return: $4 + 3 + \text{somatorio}(2)$

Terceiro return: $4 + 3 + 2 + \text{somatorio}(1)$

Quarto return: $4 + 3 + 2 + 1 = 10$

• Fatorial com Recursividade

Para calcularmos o fatorial de um inteiro **n**, basta multiplicarmos todos os números de 1 até o próprio **n**. Expressamos o fatorial de **n** por **n !**

Por exemplo:

- $4! = 4 * 3 * 2 * 1 = 24$
- $5! = 5 * 4 * 3 * 2 * 1 = 120$
- $6! = 6 * 5 * 4 * 3 * 2 * 1 = 720$

Agora note duas coisas:

$$5! = 5 * 4!$$

$$6! = 6 * 5!$$

Ou seja:

$$n! = n * (n-1)!$$

Implementando isso em código Python:

```
def fatorial(x):  
    if x==1:  
        return 1  
    else:  
        return x * fatorial(x-1)  
  
while True:  
    x = int(input("Fatorial de: "))  
    print("Fatorial: ",fatorial(x) )
```

O ponto de parada é quando a função recebe 1 como argumento.
Aí ela retorna 1.

Se não for 1, retorna o valor do argumento **x** multiplicado pelo fatorial de **(x-1)!**

Exercícios de Funções

1. Escreva um script que pergunta ao usuário se ele deseja converter uma temperatura de grau Celsius para Farenheit ou vice-versa. Para cada opção, crie uma função. Crie uma terceira, que é um menu para o usuário escolher a opção desejada, onde esse menu chama a função de conversão correta.
2. Faça um programa, com uma função que necessite de três argumentos, e que forneça a soma desses três argumentos através de uma função. Seu script também deve fornecer a média dos três números, através de uma segunda função que chama a primeira.
3. Faça um programa que recebe três números do usuário, e identifica o maior através de uma função e o menor número através de outra função.
4. A probabilidade de dar um valor em um dado é $1/6$ (uma em 6). Faça um script em Python que simule 1 milhão de lançamentos de dados e mostre a frequência que deu para cada número.
5. A série de Fibonacci é uma sequência de números, cujos dois primeiros são 0 e 1. O termo seguinte da sequência é obtido somando os dois anteriores. Faça uma script em Python que solicite um inteiro positivo ao usuário, n . Então uma função exibe todos os termos da sequência até o n -ésimo termo. Use recursividade.
6. Crie uma função que recebe um inteiro positivo e teste para saber se ele é primo ou não. Faça um script que recebe um inteiro n e mostra todos os primos, de 1 até n .
7. Um número é dito **perfeito** quando ele é igual a soma de seus fatores. Por exemplo, os fatores de 6 são 1, 2 e 3 (ou seja, podemos dividir 6 por 1, por 2 e por 3) e $6=1+2+3$, logo 6 é um número perfeito. Escreva uma função que recebe um inteiro e dizer se é perfeito ou não. Em outra função, peça um inteiro n e mostre todos os números perfeitos até n .

8. Faça um programa para imprimir:

1

2 2


```
3 3 3
```

```
.....
```

```
n n n n n n ... n
```

para um n informado pelo usuário. Use uma função que receba um valor n inteiro e imprima até a n-ésima linha.

9. Faça um programa para imprimir:

```
1
```

```
1 2
```

```
1 2 3
```

```
.....
```

```
1 2 3 ... n
```

para um n informado pelo usuário. Use uma função que receba um valor n inteiro imprima até a n-ésima linha.

10. Faça um programa, com uma função que necessite de um argumento. A função retorna o valor de caractere 'P', se seu argumento for positivo, e 'N', se seu argumento for zero ou negativo.

11. Faça um programa com uma função chamada somaImposto. A função possui dois parâmetros formais: taxaImposto, que é a quantia de imposto sobre vendas expressa em porcentagem e custo, que é o custo de um item antes do imposto. A função “altera” o valor de custo para incluir o imposto sobre vendas.

12. Faça um programa que converta da notação de 24 horas para a notação de 12 horas. Por exemplo, o programa deve converter 14:25 em 2:25 P.M. A entrada é dada em dois inteiros. Deve haver pelo menos duas funções: uma para fazer a conversão e uma para a saída. Registre a informação A.M./P.M. como um valor 'A' para A.M. e 'P' para P.M. Assim, a função para efetuar as conversões terá um parâmetro formal para registrar se é A.M. ou P.M. Inclua um loop que permita que o usuário repita esse cálculo para novos valores de entrada todas as vezes que desejar.

13. Faça uma função que informe a quantidade de dígitos de um determinado número inteiro informado.

Soluções

1.

Vamos usar as variáveis **C** para representar a medida do grau em Celsius, e a variável **F** para representar em Farenheit.

A fórmula para conversar de Celsius para Farenheit é:

$$F = C \cdot \frac{9}{5} + 32$$

Vamos chamar de **C_para_F(C)** uma função que recebe o grau Celsius, **C**, e retorna o valor em Farenheit. O código Python da fórmula acima é, então:

```
def C_para_F(C):  
    F = (C*9/5) + 32  
    return F
```

Agora o contrário. Vamos chamar de **F_para_C(F)** uma função que recebe graus Farenheit **F** como argumento e retorna o valor convertido para Celsius. A fórmula Matemática é:

$$C = (F - 32) \cdot \frac{5}{9}$$

O código Python da função é:

```
def F_para_C(F):  
    C = (F-32)*5/9  
    return C
```

Por fim, a função **menu()** que pergunta ao usuário que tipo de conversão ele quer fazer.

Uma vez que ele optou, pedimos a medida que ele deseja converter e por fim usamos IF e ELSE para chamar a função correta.

Nosso código Python final fica:

```
def C_para_F(C):  
    F = (C*9/5) + 32  
    return F
```

```
def F_para_C(F):
    C = (F-32)*5/9
    return C
```

```
def menu():
    while True:
        op = int(input('1. Celsius para Farenheit: \n' +
                       '2. Farenheit para Celsius: '))

        if op==1:
            C=int( input('Graus Celsius: ') )
            print('Convertido: ', C_para_F(C), ' graus Farenheit\n')
        elif op==2:
            F=int( input('Graus Farenheit: ') )
            print('Convertido: ', F_para_C(F), ' graus Celsius\n')
        else:
            print('Opção inválida')
```

menu()

2.

Vamos chamar de **x**, **y** e **z** os três números que o usuário vai fornecer e passar para a função **soma()**.

Dentro dela, armazenamos o valor da soma na variável **res** (de resultado) e retornamos esse valor:

```
def soma(x,y,z):
    res = x + y + z
    return res
```

Agora, na função **media()**, também recebe estes três argumentos **x**, **y** e **z**.

Usamos a função **soma()** para calcular a soma deles, dentro da função

media() mesmo, sem problemas, e armazenamos o valor da soma na variável **som** que então é dividida por 3 e armazenada na variável **med** (de média), e então retornamos essa variável:

```
def media(x,y,z):
    som = soma(x,y,z)
    med = som/3
    return med
```

Por fim, criamos uma função **menu()**, com que é invocada infinitamente, que fica pedindo os três números ao usuário, mostrando a soma e a média.

Nosso código completo fica:

```
def soma(x,y,z):  
    res = x + y + z  
    return res  
  
def media(x,y,z):  
    som = soma(x,y,z)  
    med = som/3  
    return med  
  
def menu():  
    x = int(input('Primeiro numero: '))  
    y = int(input('Segundo numero : '))  
    z = int(input('Terceiro numero: '))  
  
    print('Soma: ', soma(x,y,z))  
    print('Media:', media(x,y,z))  
  
while True:  
    menu()
```

- Otimizando um código **Python**

O exemplo dado acima está bem didático, passo a passo, para melhor entendimento de vocês.

Porém, a medida que formos estudando, fazendo programas e scripts, podemos ir *enxugando* o código, ou seja, escrevendo cada vez menos, usando menos variáveis etc.

Ou seja, otimizando. Fazer a mesma coisa, mas gastando menos memória, menos processamento e menos linha, assim também economizamos tempo.

Na função **soma**, não precisamos criar a variável **res**. Podemos dar retorno direto com o valor (x+y+z).

A função **media**, ao invés de receber três argumentos, pode diretamente receber só o valor da soma.

E o argumento que passamos pra ela, da soma? Ué, chamamos a função **soma** direto:

```
media ( soma(x,y,z) )
```

E dentro dessa função **media()** não precisamos das variáveis **som** e **med**, podemos dar um retorno direto.

Nosso código, mais limpo, fica assim:

```
def soma(x,y,z):
```

```
    return (x + y + z)
```

```
def media(num):
```

```
    return num/3
```

```
def menu():
```

```
    x = int(input('Primeiro numero: '))
```

```
    y = int(input('Segundo numero : '))
```

```
    z = int(input('Terceiro numero: '))
```

```
    print('Soma: ', soma(x,y,z))
```

```
    print('Media:', media( soma(x,y,z) ))
```

```
while True:
```

```
    menu()
```

3.

- Descobrir o **maior** número

Nossa função **maior()** tem três parâmetros: x, y e z, que são os números que iremos passar como argumento.

Dentro da função, definimos a variável **max** que recebe o valor de x. Vamos assumir que é o valor máximo.

Comparamos, via **Teste incondicional IF**, **max** com **y**.

Se **y** for maior que **max**, fazemos **max** receber o valor de **y**.

Veja que **max** vai armazenar o maior valor, dentre x e y.

Em seguida, fazemos a mesma coisa, comparando **max** com **z**.

E prontinho, **max** vai ter o maior valor dentre x, y e z.

Agora é só dar um *return*.

- Descobrir o **menor** número

A lógica é absolutamente a mesma da anterior.

Vamos usar uma variável **min** que vai ser encarregada de armazenar o menor valor.

Inicialmente, fazemos receber o valor de **x**. Depois comparamos **min** com **y** e depois com **z**.

Se forem menor que **min**, o valor de **min** recebe esse novo valor, pois é menor.

Por fim, basta dar um *return min* na função **menor()**.

Nosso código fica:

```
def maior(x,y,z):  
    max = x
```

```
    if y > max:  
        max = y
```

```
    if z > max:  
        max = z
```

```
    return max
```

```
def menor(x,y,z):  
    min = x
```

```
    if y < min:  
        min = y
```

```
    if z < min:  
        min = z
```

```
    return min
```

```
def menu():
```

```
    x = int(input('Primeiro numero: '))
```

```
    y = int(input('Segundo numero : '))
```

```
    z = int(input('Terceiro: numero: '))
```

```
    print("Maior: ", maior(x,y,z))
```

```
    print("Menor: ", menor(x,y,z))
```

```
print()
```

```
while True:  
    menu()
```

4.

Para gerar nossos números aleatórios de 1 até 6, vamos definir a função **gera()** que não tem nenhum parâmetro e vai, obviamente, retornar número inteiro de 1 até 6.

Para isso, vamos usar a função **randint** da biblioteca random.
Se não leu nosso tutorial, veja: [Como gerar números aleatórios em Python](#)

Essa função recebe os argumentos **1** e **6**, para gerar inteiros de 1 até 6. E para usar ela, temos que importar o módulo **random**.
Ela fica:

```
import random
```

```
def gera():  
    return random.randint(1,6)
```

Agora, vamos criar a função **repete(n)**, que vai repetir **n** lançamentos de dados, onde **n** o usuário vai escolher quantas vezes vai jogar os dados.

Vamos armazenar os resultados 1, na variável *test1*.

Vamos armazenar os resultados 2, na variável *test2*.

...

Vamos armazenar os resultados 6, na variável *test6*.

Então inicializamos essa variável com todas elas igual a 0.

Agora vamos fazer um laço **for** pra rodar **n** vezes o lançamento de dados. Cada vez que lança o dado (simplesmente chamando a função **gera()** - uma maravilha função né?), armazenamos na variável **test**.

Em seguida, vamos fazer várias comparações com IF, ELIF e ELSE, para saber se o valor do dado é 1, 2, 3, 4, 5 ou 6. E quando achar o valor certo, incrementar em 1 o valor da variável *test1*, ou *test2* ou ... *test6*.

Ou seja, ao final do laço **for**, as variáveis test1, test2, ..., test6 terão quando vezes foi sorteado o número 1, 2, 3, 4, 5 e 6 respectivamente.

Então simplesmente damos seis prints, para exibir quantas vezes cada número saiu, bem como mostramos a porcentagem de frequência de cada valor.

Veja o código:

```
import random
```

```
def gera():  
    return random.randint(1,6)
```

```
def repete(n):  
    test1=test2=test3= \  
    test4=test5=test6 = 0  
    for val in range(n):  
        test = gera()
```

```
        if(test==1):  
            test1 += 1  
        elif(test==2):  
            test2 += 1  
        elif(test==3):  
            test3 += 1  
        elif(test==4):  
            test4 += 1  
        elif(test==5):  
            test5 += 1  
        else:  
            test6 += 1
```

```
print("Numero 1 saiu ", test1, " vezes = ", (test1/n)*100, " %")  
print("Numero 2 saiu ", test2, " vezes = ", (test2/n)*100, " %")  
print("Numero 3 saiu ", test3, " vezes = ", (test3/n)*100, " %")  
print("Numero 4 saiu ", test4, " vezes = ", (test4/n)*100, " %")  
print("Numero 5 saiu ", test5, " vezes = ", (test5/n)*100, " %")  
print("Numero 6 saiu ", test6, " vezes = ", (test6/n)*100, " %")
```

```
def menu():
```



```
n = int(input('Quantos lançamentos de dado? '))  
repete(n)
```

```
while True:  
    menu()
```

PS 1: Quanto maior for o n , mais a porcentagem se aproxima de 16,67% pois vai tendendo à probabilidade, que é de $1/6$.

PS 2: Mais na frente, quantos estudarmos sequências, você aprenderá maneiras mais simples de se trabalhar com várias variáveis de uma vez só.

5.

Vamos criar uma função chamada **fibonacci(n)** que tem o parâmetro n . O argumento que você deve passar para esta função é um inteiro positivo, maior ou igual a 2.

Como a série de Fibonacci é formada somando seus dois termos anteriores, sua fórmula geral é:

$$\text{fibonacci}(n) = \text{fibonacci}(n-1) + \text{fibonacci}(n-2)$$

Mas como toda boa **função que usa recursividade**, ele tem que ter um *stop*, pois uma hora ela vai ter que parar. Fazemos isso usando testes condicionais **IF**.

No nosso caso, vai ter dois *stops*.

Se o valor de n for igual 1, vamos retornar 0 (pois o primeiro termo é 0).

Se o valor de n for igual 2, vamos retornar 1 (pois o segundo termo é 1).

Pronto, para valores maiores que 2, basta somarmos os dois termos anteriores.

Agora vamos montar nossa função **menu()**.

Ela pede o termo n ao usuário, que deve ser inteiro e maior que 2.

Agora ela deve imprimir na tela os valores de fibonacci(1), fibonacci(2), fibonacci(3)....até fibonacci(n).

Lembre-se, nossa função mostra o n -ésimo termo!

Então temos que imprimir todos, de 1 até n .

Fazemos isso usando um laço **for**, que vai de 1 até n (**função range**).

E prontinho, vai exibir cada termo da sequência de Fibonacci, um por linha!

Danada essa função recursiva, não?

Nosso código ficou assim:

```
def fibo(n):  
    if n==1:  
        return 0  
    elif n==2:  
        return 1  
    else:  
        return fibo(n-1) + fibo(n-2)  
  
def menu():  
    n = int(input('Exibir ate o termo (maior que 2): '))  
  
    for val in range(1,n+1):  
        print(fibo(val))  
  
while True:  
    menu()
```

6.

Primeiro, vamos criar uma função chamada **primo(n)** que recebe um inteiro positivo n como parâmetro.

Devemos testar esse valor n para saber se ele é primo ou não.

Relembrando: número primo é aquele que é divisível somente por 1 e por ele mesmo (por -1 e -ele também).

Ou seja, o resto da divisão por 2, 3, 4..., até $n-1$ tem que ser **diferente de 0**. Lembra do **operador % - resto da divisão** ?

Pois é, foi usado pra saber se um número é par ou não, agora vamos usar para detectar um número primo.

Dentro da **primo()** simplesmente saímos calculando o resto da divisão de n por 2, por 3, ...até **$n-1$** .

Se der 0, fazemos *return False*, pra dizer que o número não é primo.

Se após todos esses testes do laço **for** ele não der *False*, é porque é verdade esse bilete...digo, ele é primo e damos um *return True*.

- Código dessa função:

```
def primo(n):  
    for val in range(2,n):  
        if n % val == 0:  
            return False  
  
    return True
```

- Exibindo números primos em um intervalo

Ok, já sabemos testar se um número é primo ou não.

Agora precisamos exibir todos os primos de 1 até n .

Vamos fazer isso através da função **exibe()**, que vai pedir um inteiro positivo maior que 1 para o usuário.

A seguir, vamos criar um laço *for* que vai testar os números 2, 3, 4, ... até o n , pra saber se cada um desses valores é primo.

Se for, imprime ele. Fazemos isso usando a função `range`:

```
def exhibe():  
    n = int(input('Exibir primos até o número: '))  
    for val in range(2,n+1):  
        if(primo(val)):  
            print(val)
```

Nosso script que descobre e exhibe os números primos é, portanto:

```
def primo(n):  
    for val in range(2,n):  
        if n % val == 0:  
            return False  
  
    return True
```

```
def exhibe():  
    n = int(input('Exibir primos até o número: '))  
    for val in range(2,n+1):
```

```
if(primo(val)):
    print(val)
```

```
while True:
    exhibe()
```

- Exercício proposto:

Otimize o código acima de duas maneiras:

1. Faça a função `primo()` testar os valores até **$n/2$** ao invés de **n**
2. Faça a função `primo()` testar até a raiz quadrada de n

Primeiro range: `range(2,int(n/2)+1)`

Segundo range: `range(2,int(math.sqrt(n))+1)`

No segundo caso, escreva lá no início do script: **`import math`**, para importar o módulo *math*. Vamos aprender no próximo tutorial sobre módulos.

Também passamos os valores para inteiro, com a função **`int()`**, por o range só aceita valores inteiro.

7.

Temos que fatorar o número para saber se ele é um número perfeito. Para isso, vamos usar o operador `%` de resto da divisão.

Na função **`perfeito(n)`**, devemos pegar o argumento passado n e fazer o resto da divisão por 1, 2, 3, 4, ..., até n . Fazemos isso com a função **`range(1, n+1)`**, que vai testar de 1 até n .

Se der 0 o resto da divisão, é porque esse valor é um divisor de n . Vamos somar esse valor a uma variável chamada *soma*, que inicializamos com valor 0, no início da função **`perfeito()`**.

No final, pegamos o valor contido em *soma* e comparamos com n . Se forem iguais, retornamos *True*, pois é perfeito. Se não forem iguais, não é um número perfeito e retornamos *False*.

- Exibindo números perfeitos

Agora vamos para a função **exibe()**, que inicialmente pede um número inteiro ao usuário, **n**,

O que essa função vai fazer é jogar de 1 até **n** na função **perfeito()** e exibir os números que retornarem *True*, ou seja, que são perfeitos!

Fazemos isso com um simples laço *for* com `range(1,n+1)` e um **IF** dentro do laço, para testar o retorno da função **perfeito()**.

Teste o código abaixo com os seguintes valores: 28, 496, e 8128. São perfeitos?

- Código **Python**

```
def perfeito(n):  
    soma=0  
    for val in range(1,n):  
        if n % val == 0:  
            soma += val  
  
    if soma==n:  
        return True  
    else:  
        return False  
  
def exhibe():  
    n = int(input('Exibir perfeitos até o número: '))  
  
    for val in range(1,n+1):  
        if(perfeito(val)):  
            print(val)  
  
while True:  
    exhibe()
```

8.

O usuário vai fornecer o valor *n*, nas função **exibe()**.
Note que temos *n* linhas impressas.

Na primeira linha, o número 1 aparece uma vez.
Na segunda linha, o número 2 aparece duas vezes.
...
Na n -ésima linha, o número n aparece n vezes.

O grande segredo aqui é: usar um laço *for* dentro de outro laço *for*.
O primeiro *for* fica responsável por imprimir as linhas, ou seja, ele imprime n linhas, de 1 até $n+1$ através da função **range(1,n+1)** e a variável temporária desse *for* é a *lin* (de linha).

Agora vem o pulo do gato.
Dentro desse *for*, vamos colocar outro *for*.

Se situe: agora estamos dentro de cada uma das linhas.
Dentro de cada uma delas, vamos repetir o número *lin* (que representa a linha) *lin* vezes. Isso é feito com a função **range(lin)**.

Dentro dessa segundo *for*, vamos imprimir o caractere **lin** (número da linha).
Vamos reservar um espaço de quatro inteiros, através do comando “**%4d**”, que vai imprimir o que tiver depois do operador %.

Como não queremos quebra de linha agora, ao final da função *print*, colocamos um **end=”** para sinalizar que não faça nada ao final da string.

Quando terminar esse laço *for* interno, aí sim, damos um *print* pra quebrar a linha.

Nosso código fica assim:

```
def desenho(n):  
    # Primeiro, o número de linhas  
    for lin in range(1,n+1):  
        #Dentro de cada linha, vamos repetir 'lin' vezes o número  
        for col in range(lin):  
            print("%4d" % lin, end="")  
        print()  
  
def exibe():  
    n = int(input('Valor de n: '))  
    desenho(n)
```

```
while True:
    exibe()
```

9.

Se, na questão anterior, a gente repetia o número da linha várias vezes:

1

2 2

3 3 3

Agora, vamos imprimir o número da coluna:

1

1 2

1 2 3

Basta uma pequena alteração, veja:

```
def desenho(n):
    # Primeiro, o número de linhas
    for lin in range(1, n+1):
        # Dentro de cada linha, vamos imprimir o
        # o número da coluna, dentro de cada linha
        for col in range(1, lin+1):
            print("%4d" % col, end="")
        print()
```

```
def exibe():
    n = int(input('Valor de n: '))
    desenho(n)
```

```
while True:
    exibe()
```

13.

Uma maneira bem simples de calcular quantos dígitos tem um número, é transformar ele em string.

Se você tem um número **n** e quiser transformar ele em *string*, basta usar a função **str()**

- **str(n)** -> retorna uma string

Agora, tudo que temos fazer é calcular o tamanho dessa string.

Isso é facilmente feito com a função **len()**

Se *texto* é uma string, a função **len(texto)**, retorna o tamanho dessa string.

Agora, só precisamos jogar dentro de **len()** o número que transformamos em string (**str(n)** - onde *n* é um inteiro).

Código Python do Script

```
def contaDigito(n):  
    return len( str(n) )  
def exhibe():  
    n = int(input('Forneça um inteiro: '))  
    print(contaDigito(n), ' dígitos')  
  
while True:  
    exhibe()
```


Módulos

Neste tutorial, vamos dar início ao nosso estudo sobre **módulos em Python**, que é uma maneira de organizar nossos projetos e programas.

Aqui, vamos dar uma pincelada teórica sobre o que é um módulo, para que serve, onde e como é usado etc. No próximo tutorial, vamos criar um exemplo prático e real de um projeto usando módulos.

Módulo em Python - O que é? Para que serve? Como funciona? Como se usa ?

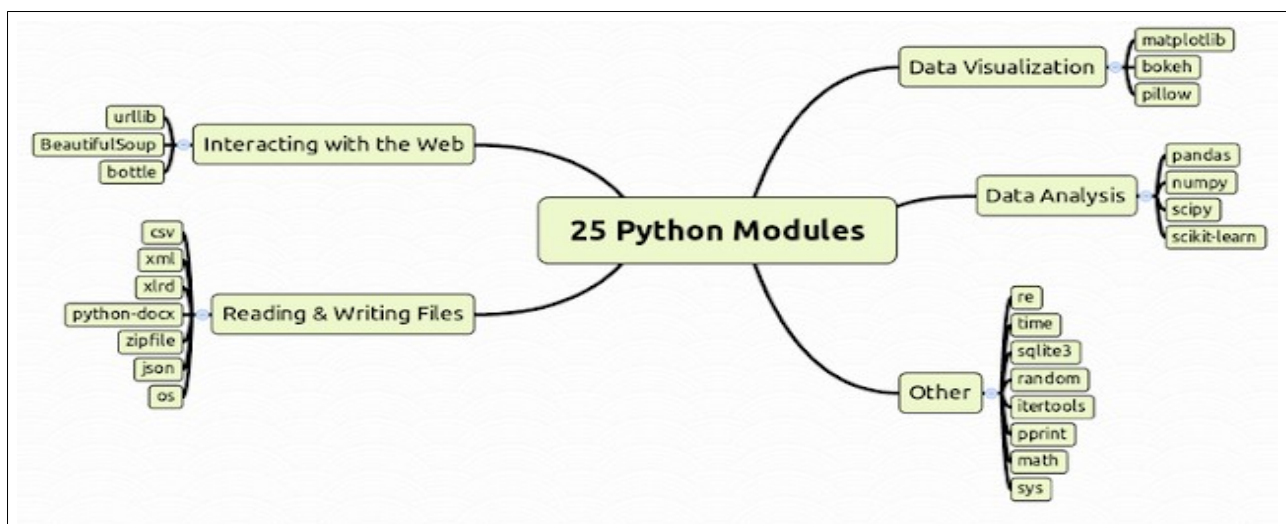
- **Módulo - O que é?**

De uma maneira bem simples, clara e objetiva: módulo é um arquivo. Um arquivo que contém código Python.

Exemplo de módulos:

- `math.py`: para trabalhar com Matemática
- `random.py`: módulo para se trabalhar com números aleatórios
- `os.py`: módulo para se trabalhar com arquivos
- `time.py`: para trabalhar com tempo (data, dia, ano, etc)

Veja alguns módulos que já vem no Python, por padrão:



Dentro desses arquivos, existem muitas, mas muitas funções mesmo, dos mais diversos tipos e dos mais variados propósitos.

Módulo - Para que serve ?

Basicamente, para organizar seus projetos.

Até o momento, em nosso **Curso de Python**, fizemos scripts de 10, 15, 20 linhas no máximo. Mas isso se deve, principalmente, por termos feitos tarefas simples.

Agora, imagine projetos como Instagram e Youtube, que usam Python. Agora, imagine um grande programa, um sistema bem robusto...facilmente passa de centenas e até milhares de linhas de código.

Pense agora no trabalho que daria para escrever milhares de linha de código em um único arquivo **.py**? Não dá, ficaria muito desorganizado.

E é aí que entra o conceito de **módulo**, a ideia basicamente é dividir o código em 'pedaços' menores, arquivos **.py**, de assuntos específicos e relacionados entre site

Módulo - Onde é usado ?

Imagine que você e sua empresa foram contratados para desenvolver um jogo.

Um jogo grande, complexo, como LoL ou WoW. Será necessário uma equipe muito grande de profissionais.

Por exemplo, alguém responsável por desenhar os personagens e cenários. Outra pessoa pra ficar responsável pelo áudio, outro pela lógica do jogo, alguém no banco de dados, outro para tratar a parte *online* do jogo.

Agora imagine tudo isso, num bloco só de código.

Não dá.

- **Módulo `menu.py`**

Ao iniciar seu jogo, deve aparece o menu para escolher modo de jogo, mapa, personagens etc. Tudo isso deve ficar no módulo **menu.py**, que vai ficar responsável por exibir os menus de opções.

Por exemplo, para aparecer o menu inicial, damos o comando:

menu.iniciar()

Ou seja, a função **iniciar()** do módulo **menu.py** é executada.

Depois, o menu de personagens:

menu.personagens()

Agora, os mapas: **menu.mapas()**

Ou seja, basta colocarmos um ponto . e em seguida rodar a função correta.

- **Módulo **audio.py****

A solução é simples, usar módulo.

Pegue a equipe de sonoplastia, e deixem eles criarem o módulo **audio.py**, que vai ter uma porção de funções.

Por exemplo, para rodar o som do mar, basta dar o comando:

audio.som_mar()

Pronto, foi invocada a função **som_mar()**, dentro do módulo **audio.py**

- **Módulo **video.py****

Você vai precisar de uma boa equipe responsável pela qualidade visual do game.

Camadas, detalhes dos personagens, do mapa etc.

Um bom grupo de programadores vai se dedicar única e exclusivamente a parte gráfica.

Mas eles te explicaram todas as funções que o módulo **video.py** tem, então para detectar a resolução correta, basta rodarmos a função **resolucao()** do módulo **video.py**

Fazemos isso assim:

video.resolucao()

E prontinho, a resolução é detectada e carregada corretamente.

- **Módulo `logica.py`**

Um personagem da infantaria, com espada e escudo é lento, porém as flechas dos arqueiros ferem ele muito pouco.

Porém se uma catapulta atirar em sua direção, muitos morrem.

Talvez seja interessante mandar a cavalaria ir na frente, destruir essas catapultas, mas tendo cuidado para não serem pegos pelos arqueiros, que atiram de longe.

Pessoal, isso tudo é lógica. Envolve Matemática (escudo aguenta 4 golpes por segundo, ele dá o golpe de espada a cada 2s, se você evoluir seu treinamento, ele luta mais rápido e desfere mais fortemente os golpes), vários cálculos.

Deixa que a equipe de lógica faz essa parte do código e coloca tudo no módulo **`logica.py`**, então para iniciar uma batalha, basta rodar:
`logica.batalha_iniciar()`

Ficou claro a função dos módulos?

São arquivos com trechos de códigos relacionados, organizam nosso projeto, fica mais fácil de criar programas maiores.

No tutorial seguinte, vamos te ensinar como criar e usar melhor seus módulos Python.

import - Como criar, importar e usar um Módulo em Python

Neste tutorial, vamos aprender a criar, importar e usar um módulo, através da *keyword* **import**.

- **Como Criar um Módulo**

Vamos criar um módulo chamado **calculadora.py**, ou seja, um arquivo com esse nome e extensão.

Nele, vamos definir 4 funções:

- 1.soma(x,y)
- 2.subtracao(x,y)
- 3.multiplicacao(x,y)
- 4.divisao(x,y)

Código que já fizemos várias e várias vezes durante nosso **Curso de Python na web**.

Uma dica muitíssimo importante, para quando formos criar módulos, é comentar seu código.

Antes de toda função de um módulo, adicione um comentário sobre a função, dizendo o que ela faz, os parâmetros e seu retorno.

Isso ajuda outros programadores que forem usar seus módulos e você também. Nosso módulo **calculadora.py** ficará assim:

calculadora.py

Esse módulo realiza as 4 operações matemáticas

Recebe dois números e retorna a soma

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

Recebe dois números e retorna a diferença

```
def subtracao(x,y):  
    return x-y
```

Recebe dois números e retorna o produto

```
def multiplicacao(x,y):
```

```
return x*y
```

```
# Recebe dois números e retorna a divisão do primeiro pelo segundo  
def divisao(x,y):  
    return x/y
```

- **Como Importar um Módulo: `import`**

Colocou o código anterior em um arquivo chamado **calculadora.py**? Agora salve esse arquivo, é seu módulo, seu código organizado, com funções prontas para serem usadas.

Agora abra um novo arquivo Python.

Para poder usar o módulo que criou, você vai usar a *keyword* **import** junto do nome do seu módulo (sem a extensão `.py`), bem no começo do código:

```
import calculadora
```

Seu arquivo `calculadora.py` deve estar na mesma pasta deste script que está programando, ok? E prontinho, para importar um módulo é só isso.

Ao fazer isso, é como se você estivesse 'jogando' todo o código de **calculadora.py** dentro desse seu novo script. Tudo que tá dentro do módulo, você vai poder usar como se estivesse direto no seu script.

- **Como Usar um Módulo**

Agora que já criamos e importamos o módulo, precisamos aprender como usar as funções contidas naquele arquivo.

Quando queremos chamar uma função **bar()**, apenas digitamos: **bar()** e pronto, ela roda, pois está naquele arquivo que você está escrevendo.

Mas se a função estiver em outro arquivo, ou seja, em um módulo, de nome **fu.py** você deve digitar:
fu.bar()

Ou seja, primeiro o nome do módulo, um ponto e o nome da função, ok?
Vamos agora criar um programa que importa esse módulo e usa suas funções.

```
import calculadora

while True:
    print("1. Soma")
    print("2. Subtração")
    print("3. Multiplicação")
    print("4. Divisão")
    op=int(input("Que operação deseja realizar: "))
    x=float(input("Primeiro numero: "))
    y=float(input("Segundo numero: "))

    if op==1:
        print("Soma:", calculadora.soma(x,y))
    elif op==2:
        print("Subtração:", calculadora.subtracao(x,y))
    elif op==3:
        print("Multiplicação:", calculadora.multiplicacao(x,y))
    elif op==4:
        print("Divisão:", calculadora.divisao(x,y))
    else:
        print("Opção inválida,tente novamente")
```

Simples, não?

Nunca mais precisaremos escrever essas funções novamente, sempre que precisarmos usar elas, bastar dar um **import** nesse módulo e usar diretamente suas funções.

Quando um programador ou empresa vai iniciar seus projetos, eles nunca partem do zero. Quando vamos iniciar um novo programa, é certa já termos algum código pronto, que pode ser reutilizado.

Por isso é importante salvar seus códigos, seus módulos, comentar, deixá-los bem organizados e fáceis de se ler e entender, pois não tenha dúvidas que vai usá-los novamente no futuro.

- **Importar Função de um Módulo:** **from** modulo**import** function

Quando fizemos:

import calculadora

Importamos tudo, todo o código, todas as funções do módulo calculadora.py. Porém, alguns módulos que usaremos são simplesmente gigantes, com milhares e milhares de linhas.

Muitas vezes queremos utilizar apenas uma ou outra função desse módulo, e acabamos dando um *import* e importamos milhares de linhas de código para no script, sem necessidade alguma.

Isso deixa nosso programa mais lento e pesado.

Por isso, é melhor importarmos somente algumas funções específicas.

Para importarmos uma função **function()** de um módulo de nome **modulo**, basta fazermos

•**from** modulo **import** function

Prontinho.

Agora é só usar **function()** direto (nem precisa fazer modulo.function()).

Como Gerar Números Aleatórios em Python - A Biblioteca random

Neste tutorial, vamos aprender como gerar números aleatórios usando o módulo **random** da biblioteca padrão do Python.

- **Números Aleatórios - Por que usar ?**

A utilidade de se gerar números aleatórios, são enormes.

Um exemplo bem claro e muito utilizado, são em jogos.

Rolar os dados, escolher inimigos e personagens aleatoriamente, localização nos mapas, alguns games ficam 'sorteando' coisas (skins, gemas, poder, equipamentos etc), tudo isso através de números aleatórios.

Já perdeu a senha ou fez algum cadastro que necessitava receber um código numérico ou alfanumérico (tipo, 23&6342h) em seu e-mail ou via SMS? Pois é, aquele código é gerado aleatoriamente.

Muitas vezes, para fazer simulações, principalmente científicas e de outros cunhos acadêmicos, é necessário usar uma enormidade de números aleatórios.

Se já teve uma cadeira de Probabilidade e Estatística na faculdade, sabe que se usa bastante números aleatórios.

São um dos conceitos mais importantes no mundo da computação, e vamos aprender agora a usar a biblioteca **random** (*random module*) em Python.

- **Gerar Número Inteiro Aleatório: A Função **randint****

Primeiro, vamos aprender como gerar um número, aleatoriamente, que seja um inteiro.

Para isso, vamos usar a função **randint**, da biblioteca padrão do Python, a **random**. Ou sej, bem no início, devemos importar esse módulo:
import random

Agora, vamos usar a **randint** para gerar um número.

Essa função tem dois parâmetros, vamos chamar de **inicio** e **fim**:

random.randint(inicio, fim)

Essa função retorna um número inteiro de 'inicio' até 'fim' (inclusive eles).

Por exemplo:

- **randint(1,10)**: pode gerar qualquer número do 1 até o 10
- **randint(1,1000)**: pode resultar qualquer número 1,2, 3, ..., 100

•Exemplo de uso da função randint()

Crie um programa em Python que simula o resultado de um dado, ou seja, gera números aleatórios de 1 até 6, quantas vezes o usuário desejar.

Como explicado, a primeira coisa é importar a biblioteca **random**, padrão do Python. Depois, simplesmente usar a **função randint** com 1 e 6 como argumentos, veja como fica:

```
import random
```

```
continuar=1
```

```
while continuar:
```

```
    print("Número aleatório gerado:", random.randint(1,6))
```

```
    continuar=int(input("Gerar novamente?\n1.Sim\n0.Não\n"))
```

• Definindo Melhor o Intervalo: **randrange()**

A função **randrange** é bem similar a **função range** (leia novamente esse tutorial para lembrar), exatamente a mesma maneira de se usar os parâmetros, a diferença que ela vai retornar apenas um valor.

- Um argumento:

random.randrange(x) - vai gerar um número aleatório de 0 até x-1

Por exemplo:

random.randrange(5) - Pode retornar:0, 1, 2, 3 ou 4

- Dois argumentos:

random.randrange(x,y) - vai gerar um número de x até y-1

Por exemplo:

random.randrange(1,5) - Pode sair: 1, 2, 3 ou 4

•Três argumentos:

random.randrange(x,y,z) - Pode gerar de x até y-1, mas ao invés de 1 em 1, ela 'pula' de z em z, ou seja, z é o salto.

Por exemplo:

random.randrange(0,10,2) - Vai gerar uma dos seguintes números: 0, 2, 4, 6 ou 8 (ou seja, todos os pares de 0 até 8).

Para gerar todos os números múltiplos de 3 menores que 100, fazemos:

random.randrange(0, 101, 3) : 3, 6, 9, ..., 99

- **Gerar Número Aleatório Decimal: random()**

Nos exemplos anteriores, as funções geram e recebem apenas números inteiros, ou seja, não servem para se trabalhar com números decimais.

Mas, e se a gente quiser um número quebrado aleatório?

Aí usamos a função **random()**, ela não recebe nenhum argumento.

Usou ela, então te retorna decimal de 0.0 até 1.0

Basta fazer: **random.random()** e terá seu número quebrado aleatório.

- **Definir Aleatório Flutuante: função uniform()**

Existe ainda mais uma função para gerar valores flutuantes de maneira aleatória, que é através da função **uniform()**.

Esta função recebe dois argumentos, um valor de início e outro final, podem ser valores **float** e te retorna um **float** aleatório.

Por exemplo, vamos supor que você é professor, tem que entregar a nota de 200 alunos e está totalmente sem saco.

Então você decide que vai dar notas aleatórias, de 5.0 até 10.0

Basta usar a função **uniform**:
random.uniform(5.0 , 10.0)

Pronto, ela vai te retornar uma nota rapidinho :D
Agora é curtir suas férias.

Caso tenha medo que algum aluno reclame, é só dizer 'olha, posso até corrigir sua prova, mas se eu encontrar mais erros, eu abaxo sua nota).

Exercício Proposto

Crie um script em Python que gere 6 números aleatórios para a Mega-Sena (neste jogo, você pode escolher dezenas de 1 até 60). Não pode repetir dezena.

O Módulo math - Matemática no Python

Lembra quando explicamos a utilidade dos módulos?

De organizar nosso código e prover a reutilização de código sem precisar programar tudo de novo?

Pois é, o módulo **math** é o melhor exemplo disso. É possível fazer muuuuita coisa da Matemática simplesmente usando as funções prontas dessa biblioteca.

Para usar em seus programas, basta importar:

import math

Para conhecer mais as funções e possibilidades desse módulo, acesse a documentação oficial do Python:

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

Vamos agora fazer um resumo e falar das principais funções do módulo **math**.

- **Trigonometria na *math***

Há várias funções que fazem cálculos de trigonometria, na **math**:

- **cos(x)** - dá o valor cosseno de x radianos
- **sin(x)** - valor do seno de x radianos
- **tan(x)** - valor da tangente de x radianos
- **acos(x)** - retorna o arco cosseno de x, em radianos
- **asin(x)** - retorna o arco seno de x, em radianos
- **atan(x)** - retorna o arco tangente de x, em radianos

- **Conversão entre graus e radianos: *degrees(x)* e *radians(x)***

Nos exemplos acima, trabalhamos com valores em radianos.

Porém, geralmente precisamos trabalhar com graus ao invés de rad.

Para transformar x radianos em graus, use a função:

degrees(x) - que ela vai te dar o valor em graus de x radianos

Para fazer o contrário, pegar um valor **x** em graus e transformar em radiano,

fazemos:

radians(x)

- **Raiz quadrada:** *sqrt(x)*

Para achar a raiz quadrada (*square root*) de um valor x , use a função **sqrt()**:
sqrt(x)

- **Constante pi:** *math.pi*

Além de funções, uma biblioteca/módulo pode armazenar constantes. Por exemplo, em geometria usamos muito o valor de ***pi*** (3.14...), por exemplo, para achar o perímetro ou área de uma circunferência, ou volume de uma esfera.

Para não ter que decorar o valor de pi ou buscar na internet, basta usar:
math.pi

É a constante pi, com várias casas decimais de precisão.

- **Logaritmo e Exponencial**

A constante **e**, base dos logaritmos naturais (número de Euler ou Néper), pode ser usada com a constante:
math.e

Já se desejar calcular uma exponenciação do tipo **e^x** (**e** elevado a **x**), use a função:

math.exp(x)

Já o logaritmo natural de **x** pode ser calculado da seguinte forma:

math.log(x)

Se preferir o logaritmo na base 10:

math.log10(x)

- **Arredondar pra baixo: *floor(x)***

Outra coisa muito utilizada, matematicamente, é o arredondamento.

Se quiser arredondar um número *x* para baixo, faça:

math.floor(x)

Por exemplo:

`math.floor(1.1) = 1`

`math.floor(3.3) = 3`

`math.floor(-1.1) = -2`

E caso o número não seja decimal, ele retorna o próprio valor:

`math.floor(10) = 10`

`math.floor(-2) = -2`

Ou seja, a função **floor()** retorna o maior inteiro que é menor ou igual a *x*.

Floor, em inglês, é chão, solo.

- **Arredondar para cima: *ceil(x)***

Já para arredondar para cima, usamos a função:

math.ceil(x)

Por exemplo:

`math.ceil(1.1) = 2`

`math.ceil(3.3) = 4`

`math.ceil(-1.1) = -1`

E caso o número não seja decimal, ele retorna o próprio valor:

`math.ceil(10) = 10`

`math.ceil(-2) = -2`

Ceiling é teto, em inglês.

A função **ceil(x)** é o menor inteiro que é maior ou igual a *x*.

Jogo em Python: Adivinhe o número

Enunciado do game:

Crie um jogo em Python onde o computador vai sortear um número de 1 até 100.

Em seguida, você vai tentar adivinhar que número foi esse.

A cada tentativa, ele vai te dizer se seu palpite foi alto, baixo ou se você acertou.

Quando acertar, deve mostrar quantas tentativas você fez até acertar.

Bom, vamos lá.

Primeiro, vamos criar nossa função **gera()**, que simplesmente vai retornar um inteiro de 1 até 100, através da função **randint()**, da biblioteca **random** que você deve importar no início do código.

Agora vamos pra função **game()**, onde a mágica vai acontecer.

Primeiro geramos o palpite do computador, e armazenamos na variável *resposta*.

Depois, inicializamos a variável *tentativa* com valor zero, ela que vai contar quantas tentativas o jogador vai fazer, até acertar.

O chute do usuário vai ser armazenado na variável *chute*, que vamos inicializar com 0.

Agora entramos no laço, que vai se repetir enquanto o jogador não acerte seu chute: ou seja, enquanto a variável *chute* seja diferente da variável *resposta*!

Como ele vai chutar, já incrementamos a variável *tentativa* em uma unidade. Em seguida, pedimos o chute dele.

Agora vamos tratar o chute com [testes condicionais IF, ELIF e ELSE](#)

Se o chute dele for maior que a resposta, dizemos que o número sorteado é menor.

Se for um chute baixo, dizemos que o sorteado é maior.
Por fim, se acertar, damos uma mensagem de parabéns, mostramos o número sorteado bem como quantas tentativas a pessoa teve até acertar.

Prontinho.

Agora colocamos um laço *while* pra chamar direto a função **game()** e pode jogar a vontade!

- **Código Python do Jogo**

```
import random
def gera():
    return random.randint(1,100)

def game():
    resposta = gera()
    tentativa = 0
    print("\nPalpite gerado!")

    chute=0
    while chute is not resposta:
        tentativa +=1
        chute = int(input("Qual seu chute: "))
        if chute > resposta:
            print("Errou! É um valor menor que ", chute)
        elif chute < resposta:
            print("Errou! É um valor maior que ", chute)
        else:
            print("Parabéns! O número gerado foi ",resposta, \
                  "Acertou em ",tentativa," tentativas!")

    while True:
        game()
```

Sequências

Nesta seção de nosso **Curso de Python**, vamos estudar sequências, como as *listas* e *tuplas*. Um outro tipo de sequência, que são as *strings*, já estudamos um pouquinho e vamos nos aprofundar mais na frente.

Além das sequências, vamos estudar os *dicionários*, que são usados para mapeamento *chave-valor* de dados (não esqueite, você vai entender melhor todas essas palavras).

Iremos aprender o que são, para que servem, onde devemos usar, como criar, inicializar e trabalhar com estes tipos de sequências.

Estudaremos conceitos importantes, como de busca e classificação de informações nas sequências, além de aprender como trabalhar e passar listas para funções. Vamos literalmente dissecar os principais métodos existentes nas listas e dicionários.

Mais na frente, aprofundaremos nossos conhecimentos em estruturas de dados quando estudarmos listas encadeadas, filas, pilhas e árvores.

Listas

- **Listas em Python - O que são?**

Sabe os números? Inteiros, *float*...? São um tipo de dado.

E as *strings* ? Também um tipo de dado.

E os *booleanos*, valores lógicos *True* e *False*? Tipo de dado.

Pois é, uma lista também é isso, um puro e simples tipo de dado.

Sendo mais específico, tecnicamente, uma lista é um objeto de coleção ordenada.

Mas não gosto muito dessas definições técnicas, mais confundem que ajudam.

vamos para a prática.

- **Listas em Python - Para que serve?**

Esqueça programação.

O que é uma lista?

Por exemplo, a lista de alunos de sua turma.

Lá tem o número em ordem crescente, e o nome das pessoas. Pronto, a lista é monte de informação, de maneira ordenada. Pode ser ordenada a partir do número, em ordem alfabética, por data de nascimento, por notas etc.

Agora imagine uma lista de funcionários de uma empresa.

Tem o nome deles, função, salário, tempo de empresa, pode ter outra lista com as datas para folga e férias, e por ai vai.

O fato é que é ambos exemplos são de coleção de dados ordenada, concorda?

Voltemos pro Python: aqui é a mesma coisa.

Uma lista é um punhado de informações (nenhuma, uma, duas ou milhões de dados), dispostos de maneira ordenada.

Assim como as listas do mundo real, as listas no Python nos permitem organizar melhor os dados, encontrar informações etc.

Por exemplo, uma lista de alunos em ordem alfabética, facilmente você vai achar o Francisco.

Pode ter 200 alunos na turma, mas você vai direto lá pela região da letra 'F' e facilmente encontra seu amigo Xiquinho.

Mesma coisa na programação, as listas vão nos permitir trabalhar com um montão de dados de uma vez, de uma maneira bem simples e poderosa.

- **Por que usar listas?**

Os propósitos são muitos, mas basicamente é pela facilidade de se lidar com coisas ordenadas. Elas, automaticamente, tem uma numeração, uma ordem lógica. Lembre-se que as coisas em computação são grandes, trilhões de bits, informações, números, dados, etc etc etc.

As listas nos fornecem uma infraestrutura para lidar com bastante informação de uma maneira bem simplificada e poderosa.

Por exemplo, as listas tem comprimento móvel, você pode ficar adicionando novos itens (como um novo cadastro de um produto de um supermercado) ou retirar (quando a empresa demitir um funcionário).

Você pode 'misturar' diversos tipos de dados em uma lista (como números, strings etc), inclusive colocar uma lista dentro de outra.

Se você já estudou outras linguagens de programação, pode notar que as listas lembram os *arrays* (em Java) ou ponteiros (em C/C++), principalmente pelo fato delas conterem, na verdade, referências para outros objetos nela.

Se não estudou, calma.

Nesta seção, vamos com bastante calma, com muitos exemplos e muitos códigos para você fixar bem os conceitos de sequências, pois são muitíssimo importantes!

No próximo tutorial, vamos aprender como criar uma lista e acessar seus itens.

Como criar uma Lista e Acessar seus dados

- **Como Criar uma Lista**

Chega de enrolar, chega de teoria, vamos colocar a mão na massa e começar a trabalhar com as danadas das listas.

O que define uma lista nada mais é que um par de colchetes: `[]`

Uma lista de animais:

`['gato', 'rato', 'elefante']`

Uma lista com notas da escola:

`[9.5 , 10.0 , 7.4]`

Cada dado dentro da lista é chamado de **item**.

Os itens são sempre separados por **vírgula**,

Experimente digitar uma lista dessas no *prompt* do Python.

Vai ver que ele retorna a própria lista.

É a mesma coisa que digitar qualquer tipo de dado. Se escrever um número e der enter, ele repete o número. O mesmo com strings, o mesmo com listas...afinal, são um tipo de dado também.

Obviamente, a maneira mais comum de usar listas, é atribuindo ela a uma variável.

```
meusBichos = ['Rex', 'Fred', 'Jamille']
```

```
notas = [ 10.0 , 8.5 , 7.8 , 9.0 ]
```

Exercício de Listas em Python

Crie um programa em Python que peça seu nome, sua idade, sua altura, seu peso e True se for casado ou False para solteiro.

*Em seguida, ele deve armazenar todas essas informações numa lista chamada **eu**. Por fim, imprima essa lista na tela.*

Primeiro, criamos 4 variáveis: nome, idade, altura e peso.

Usando o input para armazenar os dados, lembrando que *nome* é uma string, *idade* um inteiro, *altura* e *peso* são do tipo float. Sem mistério.

Perguntamos se o usuário é casado ou solteiro e armazenamos essa resposta uma variável *op*, que vai receber o *booleano* True ou False dependendo de seu estado civil.

Depois, criamos nossa lista de nome **eu**, abrindo colchetes, colocando diretamente as variáveis lá, separando seus itens por vírgula e fechando o colchete ao final.

Por fim, jogamos a variável **eu**, que é uma lista, dentro do print. Como o Python é super esperto, já detecta que é uma lista e imprime ela bem direitinho, veja:

```
Qual seu nome: xico matos
Qual sua idade: 30
Qual sua altura: 1.79
Qual seu peso: 89.5
Estado civil:
1. Casado
2. Solteiro
2
['xico matos', 30, 1.79, 89.5, False]
```

O código de nosso script Python ficou:

```
nome = input("Qual seu nome: ")
idade = int(input("Qual sua idade: "))
altura = float(input("Qual sua altura: "))
peso = float(input("Qual seu peso: "))

op= int(input("Estado civil:\n1.Casado\n2.Solteiro\n"))

if op==1:
    op = True
else:
    op = False

eu = [nome, idade, altura, peso, op]
print(eu)
```

Acessando os dados de uma Lista

Chamamos de **item** os dados contidos em uma lista. No exemplo anterior, nossa lista possui 5 itens.

Como podem ver, o primeiro é o *nome*, o segundo a *idade* e por ai vai, até o quinto, o booleano *op*.

Nada mais óbvio que acessar o primeiro pelo número 1, o segundo pelo número 2 e assim sucessivamente, concorda?

Errado! Na computação, não!

Em programação, o primeiro elemento é o de número 0.

Acessamos o primeiro item da tela, colocando esse índice entre colchetes:
`eu[0] = 'xico matos'`

Já o segundo, é o de índice 1:

`eu[1] = 30`

Como são 5 itens, o último tem índice 4 (5-1):

`eu[4] = False`

Se você tentar acessar um índice cujo item não existe na lista, vai obter uma mensagem de erro.

Exercício de Listas em Python

O print do exemplo anterior está muito feio. Entre colchete, com aspas, vírgulas...faça um print bonitinho, onde cada linha é uma informação e dizendo que informação é aquela, para ficar igual abaixo:

```
Qual seu nome: Xico Matos
Qual sua idade: 30
Qual sua altura: 1.79
Qual seu peso: 89.5
Estado civil:
1.Casado
2.Solteiro
2
Nome : Xico Matos
Idade : 30 anos
Altura: 1.79 m
Peso : 89.5 kg
Casado: False
```

Agora, ao invés de darmos simplesmente um **print(eu)**, vamos ter que quebrar os itens da lista. Vamos usar 5 *prints*, um para cada informação que vamos exibir.

Para isso, vamos ter que usar os itens separadamente, através de seus índices.

Para nome : eu[0]

Para idade : eu[1]

Para altura : eu[2]

Para peso : eu[3]

Estado civil: eu[4]

Vejam como ficou nosso script:

```
nome = input("Qual seu nome: ")
```

```
idade = int(input("Qual sua idade: "))
```

```
altura = float(input("Qual sua altura: "))
```

```
peso = float(input("Qual seu peso: "))
```

```
op= int(input("Estado civil:\n1.Casado\n2.Solteiro\n"))
```

```
if op==1:
```

```
    op = True
```

```
else:
```

```
    op = False
```

```
eu = [nome, idade, altura, peso, op]
```

```
print("Nome : ", eu[0])
```

```
print("Idade : ", eu[1], " anos")
```

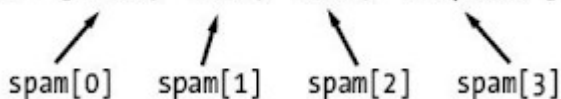
```
print("Altura: ", eu[2], "m")
```

```
print("Peso : ", eu[3], "kg")
```

```
print("Casado: ", eu[4])
```

Lembrem-se: o primeiro item de uma lista, tem sempre, **SEMPRE**, índice 0:

```
spam = ["cat", "bat", "rat", "elephant"]
```



```
      spam[0]  spam[1]  spam[2]  spam[3]
```

Nunca mais conte: 1, 2, 3, 4, 5...

Programadores começam sempre do 0: 0, 1, 2, 3, 4, 5...

Como Usar Listas em Python - Adicionar, Mudar, Concatenar e Outras Operações

Neste tutorial de nosso **curso de Python**, vamos aprender como fazer várias operações com listas, como adicionar algum item, modificar, concatenar, ver tamanho de uma lista etc.

- **Adicionar Item em uma lista: `append()`**

Vamos criar uma lista com nossas bandas favoritas.

Essa lista vai se chamar e vai ser inicialmente nula **`bandas = []`**.

Para adicionar um item na lista, vamos usar o método **`append()`**, que funciona assim:

`nomeDaLista.append(item a ser adicionado)`

Então, se quisermos adicionar a banda Iron Maiden na lista `bandas`, fazemos:

`bandas.append("Iron Maiden")`

Dê um print na lista e veja como ela está:

`print(bandas)`

```
['Iron Maiden']  
>>> |
```

- **Exercício de Listas:**

Crie um programa que pergunta se o usuário deseja adicionar uma nova banda na Lista ou exibir a lista.

Código:

`bandas = []`

`while` `True`:

`op = int(input("1. Adicionar banda\n" "2. Exibir bandas favoritas\n"))`

`if` `(op==1)`:

`banda=input("Digite nome da banda: ")`

```
bandas.append(banda)
if(op==2):
    print(bandas)
```

Teste esse código.

Se escolher a opção 1, você digita uma nova banda e insere essa string na lista **bandas** usando o método **append()**.

- **Tamanho de uma lista: len()**

Para saber o tamanho de uma lista, basta passar ela como argumento para a função **len()**.

Por exemplo, o tamanho da lista **bandas** é: **len(bandas)**

Pronto. Simplesmente isso!

- **Exercício de Lista em Python:**

Coloque mais uma opção em seu programa, de modo a exibir o tamanho atual da lista.

Nosso código fica:

```
bandas = []
```

```
while True:
    op =int(input("1. Adicionar banda\n"
                 "2. Exibir bandas favoritas\n"
                 "3. Tamanho da lista\n"))

    if(op==1):
        banda=input("Digite nome da banda: ")
        bandas.append(banda)
    if(op==2):
        print(bandas)
    if(op==3):
        print("Tamanho da lista: ", len(bandas))
```

Teste !

- **Como Mudar Item em uma Lista**

Vamos supor que temos uma lista com 3 bandas favoritas:
['Iron Maiden', 'Rush', 'Tiririca']

Você digitou Tiririca por engano, foi querer brincar e deu nisso: sua lista de bandas favoritas com Tiririca no meio.

Vamos mudar o valor de 'Tiririca' para Black Sabbath.
Para isso, o primeiro passo é saber o índice de 'Tiririca'.

Como estudamos no tutorial sobre [Como acessar os items de uma lista](#), sabemos que os índices começam com contagem 0. Assim:

- bandas[0] = 'Iron Maiden'
- bandas[1] = 'Rush'
- bandas[2] = 'Tiririca'

Ou seja, vamos alterar o índice de valor 2!
Basta fazer:

- bandas[2] = 'Black Sabbath' e prontinho, mudou o valor de um item!

Só alterar ele diretamente e fazer a mudança no valor da variável.

- **Exercício de Lista em Python**

Coloque uma opção para mudar o valor de um item. Você deve pedir a posição do índice (começando de 0) e o valor para mudar. Se o usuário escolher um valor superior ao existente, exibe uma mensagem de erro.

Nosso código agora fica:

```
bandas = []
```

```
while True:
    op = int(input("1. Adicionar banda\n"
                  "2. Exibir bandas favoritas\n"
                  "3. Tamanho da lista\n"))
```

"4. Mudar valor de item\n"))

```
if(op==1):
    banda=input("Digite nome da banda: ")
    bandas.append(banda)
if(op==2):
    print(bandas)
if(op==3):
    print("Tamanho da lista: ", len(bandas))
if(op==4):
    index=int(input("Indice que vai alterar: "))
    if(index<len(bandas)):
        temp=input("Nome da banda: ")
        bandas[index] = temp
    else:
        print("Esse indice nao existe")
```

Teste também!

- **Como Deletar Item de uma Lista: `del item`**

Uma maneira simplesmente de apagar um item, basta fazer:
`del item`

Por exemplo, se quer apagar a primeira banda da lista de favoritas:
`del bandas[0]`

E prontinho, a primeira banda some de sua lista!

- **Concatenar listas: `+`**

Concatenar nada mais é que 'unir', grudar duas listas.

Vamos supor que temos a lista:

```
bandas1 = ['Iron Maiden', 'Rush']
```

E a lista:

```
bandas2=['Deep Purple', 'Led Zeppelin']
```

E quisermos unir as duas listas, em uma só, basta 'somar' elas:
`bandas = bandas1 + bandas2`

O Python é tão esperto que vai entender e o resultado vai ser:

bandas = ['Iron Maiden', 'Rush', 'Deep Purple', 'Led Zeppelin']

Ou seja, ele **concatenou** as duas listas!

- **Repetição de listas:** *

O Python é tão esperto, mas tão esperto, que permite até você usar o símbolo de multiplicação com listas.

Se desejar repetir uma lista, basta multiplicar ela mesma por algum valor.

Vamos supor que você seja muito fã das bandas:

```
bandas1 = ['Iron Maiden', 'Rush']
```

Tão fã, que é fã 3x. Para repetir isso, basta fazer:

```
bandas = bandas * 3
```

O resultado é:

```
bandas = ['Iron Maiden', 'Rush', 'Iron Maiden', 'Rush', 'Iron Maiden', 'Rush']
```

Ou seja, repetiu ela mesma três vezes! Como se tivesse concatenado com ela mesma três vezes!

- **Operações com Listas:**

Existem diversos outros métodos (funções embutidas) nas listas, que já vem por padrão no Python.

Aconselhamos uma boa, detalhada e despretensiosa lida na documentação:

<https://docs.python.org/3/tutorial/datastructures.html>

Onde vemos que é possível:

- inverter uma lista: **reverse()**
- embaralhar uma lista: **sort()**
- apagar uma lista: **clear()**
- copiar uma lista: **copy()**
- contar quantas vezes um item aparece: **count(x)**
- retirar um elemento específico da linha: **pop(x)**

E por aí vai!

O Laço FOR com Listas em Python

Duas coisas que se casam e trabalham incrivelmente bem são: listas e laço for.

Usaremos o laço *for* para iterar a lista, ou seja, acessar item por item.

Mas, vamos ver com exemplos!

- **Exemplo 1 de laço FOR com Listas**

Faça um script que cria uma lista de 10 elementos, preenchendo eles de 0 até 9, usando laço for.

Depois, printe essa lista.

Inicialmente, criamos nossa lista **numeros**.

No começo ela está vazia.

Vamos usar um iterador, uma variável **count** que vai de 0 até 9, através da função range.

Lembramos que **count** é um número, para concatenar ele, precisamos transformá-lo antes num lista: **[count]**

A cada looping do laço *for*, concatenamos nosso número na lista, veja:

```
numeros = []
```

```
for count in range(10):  
    numeros += [count]
```

```
print(numeros)
```

- **Exemplo 2 de laço FOR e Listas**

No exemplo anterior, foi exibida uma lista no seguinte formato:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Use o laço for para exibir ela sem colchetes e vírgulas um número por linha.

Agora, ao invés de percorremos os elementos da função **range** (que retorna algo a uma lista), vamos usar nossa própria lista **numeros**.

Ou seja, o **count** vai iterar, assumir o valor de cada item da lista, um por um, e imprimimos essa variável:

```
numeros = []
```

```
for count in range(10):  
    numeros += [count]
```

```
for count in numeros:  
    print(count)
```

- **Exemplo 3 de laço FOR e Listas**

*Faça com que o script anterior também calcule a soma de todos os elementos na lista **numeros** e exiba seu resultado.*

Vamos armazenar o valor numa variável chamada **soma**, que inicializamos como 0.

A cada iteração, somamos o valor de **count** a ela, como o **count** vai receber o valor de cada item da lista, teremos a soma de todos os termos da lista:

```
numeros = []
```

```
for count in range(10):  
    numeros += [count]
```

```
soma=0
```

```
for count in numeros:  
    print(count)  
    soma += count
```

```
print("Soma: ", soma)
```

Note que nem precisamos de um novo laço *for*, usamos o laço que imprime cada item, um por linha.

- **Exemplo 4 de laço FOR e Listas**

Faça um script que exibe uma lista de 10 elementos, contados de 1 até 10. Depois, dobre cada valor dessa lista e exiba. Veja que agora são todos pares.

Preenchemos a lista com valores de 1 até 10, usando **range(1,11)**. Depois, temos que ir em termo por termo, do índice 0 até o índice 9, para dobrar os valores ali contidos.

Para acessar todos os elementos, vamos usar uma range que vai de 0 até ... o tamanho da lista, que é **len(numeros)**.

```
numeros = []
```

```
for count in range(1,11):  
    numeros += [count]
```

```
for count in range(len(numeros)):  
    numeros[count] = numeros[count]*2
```

```
for count in numeros:  
    print(count)
```

Poderíamos ter feito apenas: `numeros[count] *= 2`

- **Exemplo 5 de laço FOR e Listas**

Faça um script que exiba a uma lista que tem os números de 1 até 10, na seguinte maneira:

```
Numero 1: 1  
Numero 2: 2  
Numero 3: 3  
Numero 4: 4  
Numero 5: 5  
Numero 6: 6  
Numero 7: 7  
Numero 8: 8  
Numero 9: 9  
Numero 10: 10
```

Aqui tem uma pegadinha.

Para imprimir, vamos usar a `range(len(numeros))`, assim o laço percorre todos os índices da lista.

Porém, o primeiro índice é 0.

Se printarmos a variável **count** direto, ia aparecer:

Número 0: 1

Número 1: 2 etc

Ou seja, temos que adicionar 1 na hora de imprimir:

```
numeros = []
```

```
for count in range(1,11):  
    numeros += [count]
```

```
for count in range(len(numeros)):  
    print("Numero %2d: %2d" %(count+1, numeros[count]))
```

- **Exemplo 6 de laço FOR e Listas**

Faça um script que peça ao usuário o número de matérias da escola, ou seja, um inteiro positivo.

Em seguida, ele vai digitando o valor de cada nota, de cada matéria e isso será armazenado numa lista.

Ao final, seu script deverá fornecer a média geral do aluno.

Primeiro, perguntamos ao usuário quantas matérias ele fez provas, e armazenamos em **n**.

Depois, criamos uma lista vazia, para as notas, bem como inicializamos a variável **soma** com valor 0, ela vai somar todas as notas.

Dentro do *for*, temos que preencher **n** notas, fazemos isso com a função *range*.

Perguntamos cada nota (imprimimos **count+1**, para os índices aparecerem bonitinho para o usuário, lembre-se que para ele a contagem não começa em 0, como começa para nós programadores) e vamos adicionando cada nota como uma lista, concatenando na lista maior **notas**.

Somamos o valor de **soma** com cada nota digitada.

No final, dividimos o valor dessa soma pelo número de matérias que o aluno fez provas, e temos a média geral dela.

Nosso código Python fica:

```
n = int(input("Numero de materias: "))  
notas = []  
soma=0
```

```
for count in range(n):
    nota=float(input("Nota da materia %2d: " % (count+1)))
    notas += [nota]
    soma += nota
```

```
print("Notas: ",notas)
print("Media: %.2f" % (soma/n))
```

• Exercícios Propostos

1. Faça um script que pede ao usuário um número inteiro positivo. Em seguida, deve criar uma lista onde o primeiro termo é 0, o segundo é 1 e os outros são a sequência de Fibonacci. O número de termos é o que o usuário forneceu como número.
2. Faça um script que pede um número inteiro positivo ao usuário. Em seguida, cria uma lista com igual número de itens, onde o primeiro termo é 1!, o segundo é 2!, o terceiro é o valor de 3!, etc, até o termo que ele digitou. Ou seja, se digitou n , vai exibir até o termo de índice $n-1$, e lá na lista vai ter o valor de $(n-1)!$.

Soluções

1.

Primeiro, perguntamos ao usuário quantos termos ele deseja ver, e armazenamos isso em **n**.

Em seguida, criamos nossa lista com os termos iniciais 0 e 1, é a lista **fibo**.

Agora vamos lá, do índice 2 (terceiro elemento da sequência, já que começa em 0) até o elemento **n-1** (contando com 0, teremos n elementos).

Para cada elemento, adicionamos os dois anteriores:

```
fibo[count-1] + fibo[count-2]
```

O resultado é esse:

```
n = int(input("Gerar sequencia até o termo: "))
fibo = [0,1]
```

```
for count in range(2,n):  
    termo = fibo[count-1] + fibo[count-2]  
    fibo += [termo]
```

```
print("Sequência: ",fibo)
```

2.

Pedimos um inteiro positivo, de quantos elementos o usuário deseja ver.
Em seguida, criamos a lista, com o valor de 1 dentro, que é quanto vale 0!

Agora, entramos no for. Que vai calcular o valor do termo de índice 1, termo de índice 2, termo de índice 3...até o valor de termo $n-1$.

Para calcular cada novo termo, basta pegar o termo anterior e multiplicar pelo índice atual, afinal:

$n! = n * (n-1)!$

Nosso código fica:

```
n = int(input("Gerar sequencia até o termo: "))  
fat = [1]
```

```
for count in range(1,n):  
    termo = fat[count-1]*count  
    fat += [termo]
```

```
print("Fatoriais: ",fat)
```

Quebrando e Fatiando Listas - Operador :

Vamos iniciar nosso estudo com uma lista chamada **semana**:

```
semana=['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta', 'Sábado', 'Domingo' ]
```

Para obter uma sublista, ou seja, um pedaço dessa lista semana, vamos usar o operador dois pontos:

```
novaLista = Lista[inicio : fim]
```

Para gerar uma sublista de segunda até sexta, fazemos:

- `semana[0:5] = ['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta']`

Ou seja, começa no índice 0 e vai até o índice 4 (igual a função range).

Como queremos pegar o início da lista até o quinto item, podemos fazer assim:

- `semana[:5] = ['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta']`

Se quisermos pegar de Terça até Domingo, fazemos:

- `semana[1:] = ['Terça', 'Quarta', 'Quinta', 'Sexta', 'Sábado', 'Domingo']`

Como queremos até o final da lista, não precisamos colocar nenhum índice depois do :, pois o Python entende que você quer ir até o final da lista.

Para exibir toda a lista original, podemos fazer:

- `semana[:] = ['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta', 'Sábado', 'Domingo']`

Operador in e not in

Vamos pegar uma lista com os dias úteis da semana:

```
•semana=['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta' ]
```

Essa lista são os dias que você vai trabalhar.

Para saber se um determinado item faz parte da lista, você faz:

item in semana

Se estiver, ele retorna True.

Se não estiver, retorna False.

Rode o seguinte exemplo:

```
semana=['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta' ]
```

```
if 'Sexta' in semana:
```

```
    print("Sim, ", 'Sexta'," está na lista")
```

```
else:
```

```
    print("Não está")
```

Agora se você quiser confirmar que um item **não** está na lista, faça o teste:

item not in semana

Se não estiver, retorna True.

Se estiver, retorna False

Veja:

```
semana=['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta' ]
```

```
if 'Domingo' not in semana:
```

```
    print("Não, ", 'Domingo'," não está na lista")
```

```
else:
```

```
    print("Está")
```

Como Copiar uma Lista em Python

- **Referenciando Listas - Copiando do jeito Errado**

Vamos criar a **list1**:

```
list1 = [1, 2, 3, 4]
```

O que acontece por trás dos planos é que existe um bloco de memória no seu computador, com a lista **[1, 2, 3, 4]** armazenado lá. A variável **list1** tem um endereço que aponta para esse bloco de memória.

Agora vamos criar outra lista, a **list2**.

Fazendo uma cópia direta: **list2 = list1**

Agora faça um teste, mude um valor da **list1**, por exemplo:

```
list1[0] = 0
```

Agora a **list1** será [0,2,3,4].

Porém, se você der um print em **list2** vai ver que ela também é [0, 2, 3, 4]

```
>>> list1=[1, 2, 3, 4]
>>> list2=list1
>>> list1[0]=0
>>> list1
[0, 2, 3, 4]
>>> list2
[0, 2, 3, 4]
>>> |
```

Mas peraí Python!

Eu queria mudar só a **list1**! E não a **list2**.

Por que isso ocorre?

Simples, ao fazermos **list2=list1**, faremos com que list2 também aponte para aquele endereço de memória.

Então, se alterar o valor naquele bloco de memória, essa mudança vai ser visível para todas as variáveis que apontem para lá!

- **Copiando do Jeito Certo e Difícil**

Uma solução para copiar corretamente é copiar item por item:

```
list2 = []  
for item in list1:  
    list2.append(item)
```

- **Copiando do Jeito Certo e Fácil**

Ou uma solução ainda mais elegante:

```
list2 = [] + list1
```

O que fizemos foi concatenar a lista **list1** com outra lista (vazia), e o resultado é uma cópia de **list1**, mas agora list2 não aponta para o mesmo bloco de memória.

Bacana copiar dessa maneira, não é?

Matriz: Lista dentro de Lista

Até o momento, em nossas listas, colocamos dados como números inteiros, float e strings.

Porém, é possível colocar também uma lista, dentro de uma lista.

A lista:

- notas = [10, 9, 8]

Possui apenas três elementos, três inteiros. Uma linha e 3 colunas, ok ?

Agora vamos supor que temos três alunos, onde cada aluno possui 3 notas. Podemos inicializar uma lista com as notas desses três alunos da seguinte maneira:

```
notas= [ [10, 9, 8],  
         [9, 8, 7],  
         [8, 10, 5] ]
```

Ou seja, temos 3 listas dentro de uma lista.

Temos 3 linhas e 3 colunas (tente ver como uma matriz de elementos, uma tabela).

Como acessar elementos de uma Matriz

As notas do primeiro aluno estão na lista **notas[0]**

As notas do segundo aluno estão na lista **notas[1]**

As notas do terceiro aluno estão na lista **notas[2]**

Para acessar a primeira nota do primeiro aluno: **notas[0][0]**

Para acessar a segunda nota do primeiro aluno: **notas[0][1]**

Para acessar a terceira nota do primeiro aluno : **notas[0][2]**

Para acessar a primeira nota do segundo aluno: **notas[1][0]**

Para acessar a segunda nota do segundo aluno: **notas[1][1]**

Para acessar a terceira nota do segundo aluno : **notas[1][2]**

Para acessar a primeira nota do terceiro aluno : **notas[2][0]**

Para acessar a segunda nota do terceiro aluno : **notas[2][1]**

Para acessar a terceira nota do terceiro aluno : **notas[2][2]**

Para calcular a média das notas do primeiro aluno

$(\text{notas}[0][0] + \text{notas}[0][1] + \text{notas}[0][2]) / 3$

Para calcular a média das notas do segundo aluno

$(\text{notas}[1][0] + \text{notas}[1][1] + \text{notas}[1][2]) / 3$

Para calcular a média das notas do terceiro aluno

$(\text{notas}[2][0] + \text{notas}[2][1] + \text{notas}[2][2]) / 3$

- **Exercício de Matriz em Python**

Crie uma matriz 4 x 4 (ou seja, 4 linhas e 4 colunas, ou seja, 4 listas dentro de uma lista maior).

Popule essa lista com números de 0, 1, 2, 3 etc

Como inicializar uma matriz

Usamos um laço for para inicializar uma matriz.

Por exemplo, para inicializar uma lista de 4 elementos, todos 0, fazemos:

- `[0 for i in range(4)]`

Para inicializar uma matriz 4x4 então, basta repetirmos o código anterior dentro de outro *for*:

- `matriz = [[0 for i in range(4)] for j in range(4)]`

Inicializamos uma variável com valor 4.

Para percorrermos uma matriz, temos que usar laços aninhados, ou seja, *for* dentro de *for*.

O primeiro *for* percorre as linhas, o de dentro percorre as colunas.

Para mudarmos cada valor da matriz, a partir de 0, fazemos:

Como exibir uma matriz

Você pode imprimir uma matriz de nome **matrix** simplesmente fazendo:
`print(matrix)`

Porém, o Python vai imprimir tudo numa mesma linha, muito feio.

Para exibirmos linha por linha, vamos novamente usar o laço *for* aninhado de outro *for*.

O primeiro *for*, de cima, vai primeiro começar na primeira linha.

O laço *for* aninhado, vai imprimir cada coluna, ou seja, cada elemento da linha.

Após terminar de imprimir a linha, ou seja, quando o laço *for* de dentro termina, damos um **print()** para dar uma quebra de linha quando terminamos de imprimir a linha.

Nosso código, então, fica:

```
matriz = [ [0 for i in range(4)] for j in range(4)]  
count=0
```

```
for linha in range(4):  
    for coluna in range(4):  
        matriz[linha][coluna] = count  
        count += 1
```

```
for linha in range(4):  
    for coluna in range(4):  
        print("%4d" % matriz[linha][coluna], end="")  
    print()
```

Jogo da velha em Python

- **Lógica do jogo da velha**

Vamos usar uma [matriz \(lista dentro de lista\)](#) de tamanho 3x3 de inteiros. Um jogador vai escolher a linha e a coluna no tabuleiro, para jogar.

Se for o jogador 1, vai assinalar um X.

Se for o jogador 2, vai assinalar um O.

Porém, só pode inserir um X ou O se aquela posição estiver vazia. Cada vez que o jogador joga, deve verificar se alguma linha, coluna ou diagonal foi preenchida, terminando o jogo, avisando quem ganhou e em quantas rodadas.

- **O Tabuleiro**

Inicializamos o tabuleiro com valores iguais a 0:

```
board= [ [0,0,0],  
          [0,0,0],  
          [0,0,0] ]
```

Também podemos inicializar fazendo:

```
board = [ [0 for i in range(3)] for j in range(3)]
```

Em seguida, o código chama a função **menu()**.

- **O menu**

Criamos uma variável chamada **continuar**, com valor inicial 1.

Enquanto esse valor não for nulo, um laço *while* vai continuar perguntando se o jogador deseja jogar novamente, novamente e novamente.

- **Exibindo o tabuleiro do jogo da velha**

Na função **exibe()** vamos imprimir o tabuleiro.

Quando o jogador 1 joga, insere o número inteiro 1 em um local do tabuleiro.

Se o jogador 2 joga, insere o número inteiro -1 em um local do tabuleiro.

Para exibirmos, vamos usar dois laços *for* aninhados.
O primeiro 'cuida' da linha e o segundo cuida das colunas.

Quando nos deparamos com o valor 0 no tabuleiros, mostramos: ____
Para simbolizar que está vazio.

Se encontrar o número 1, printamos o caractere X.
Se encontrar o número -1, printamos o caractere O.

• O jogo

A variável **jogada** é que vai contabilizar o número de rodadas. Obviamente, inicia com zero.

Essa variável vai ser incrementada de um em um, a cada rodada.

Se pegarmos o resto dessa divisão dessa variável por 2, teremos valor 0 e 1 alternando.

Se somarmos 1, teremos valores 1 e 2 alternando. Vamos avisar ao usuário se é o jogador 1 ou jogador 2 através dessa lógica.

Ou seja, cada jogador é simbolizado por: $(jogada \% 2 + 1)$

Primeiro, exibimos o tabuleiro, chamando a função **exibe()**.

A seguir, pede uma linha e uma coluna.

Detalhe: o usuário vai digitar 1, 2 ou 3, mas nosso tabuleiro tem as posições 0, 1 e 2. Ou seja, quando o usuário fornecer esses números, temos que subtrair o valor de 1.

Em seguida, testamos se o valor que ele assinalou está vazio, ou seja, se tem um 0 naquela posição do tabuleiro.

Se sim, insere um 1 caso seja vez do jogador um ou -1 se for o jogador dois.

Se não for 0 a posição do tabuleiro, cai no ELSE, avisamos que não pode e subtraímos 1 da variável *jogada*, pois lá embaixo essa variável é sempre incrementada.

Após cada jogada, chamamos a função **ganhou()** dentro de um teste condicional.

Essa função checa se alguma linha, coluna ou diagonal está completa. Se sim, termina o jogo e mostra quem ganhou, em quantas rodadas, se foi linha, coluna ou diagonal que se completou, e exibimos o tabuleiro final.

Caso a função *ganhou()* retorne 0, é porque ninguém ganhou, incrementamos a variável *jogada* e o loop *while* roda novamente uma outra rodada com o outro jogador.

- **Quem ganha o jogo da velha**

Dentro da função **ganhou()**, fazemos três checagens.

Primeiro, se alguma linha tem soma de valor 3 ou -3, alguém ganhou e retornamos o valor 1.

Se alguma coluna tem soma 3 ou -3, alguém ganhou e retornamos o valor 2. Por fim, somamos as diagonais, se alguma tiver valor 3 ou -3, retornamos o valor 3.

Se passar batido por esses testes, retornamos o valor 0, pra simbolizar que ninguém ganhou.

- **Código Python do Jogo da Velha**

```
def menu():
    continuar=1
    while continuar:
        continuar = int(input("0. Sair \n"+
                               "1. Jogar novamente\n"))
        if continuar:
            game()
        else:
            print("Saindo...")

def game():
    jogada=0

    while ganhou() == 0:
        print("\nJogador ", jogada%2 + 1)
        exibe()
```

```
linha = int(input("\nLinha :"))
coluna = int(input("Coluna:"))
```

```
if board[linha-1][coluna-1] == 0:
    if(jogada%2+1)==1:
        board[linha-1][coluna-1]=1
    else:
        board[linha-1][coluna-1]=-1
else:
    print("Nao esta vazio")
    jogada -=1
```

```
if ganhou():
    print("Jogador ",jogada%2 + 1," ganhou apos ", jogada+1," rodadas")

jogada +=1
```

```
def ganhou():
    #checando linhas
    for i in range(3):
        soma = board[i][0]+board[i][1]+board[i][2]
        if soma==3 or soma ==-3:
            return 1
```

```
    #checando colunas
    for i in range(3):
        soma = board[0][i]+board[1][i]+board[2][i]
        if soma==3 or soma ==-3:
            return 1
```

```
    #checando diagonais
    diagonal1 = board[0][0]+board[1][1]+board[2][2]
    diagonal2 = board[0][2]+board[1][1]+board[2][0]
    if diagonal1==3 or diagonal1==-3 or diagonal2==3 or diagonal2==-3:
        return 1
```

```
    return 0
```

```
def exibe():
    for i in range(3):
        for j in range(3):
            if board[i][j] == 0:
                print(" _ ", end=' ')
            elif board[i][j] == 1:
                print(" X ", end=' ')
```

```
elif board[i][j] == -1:  
    print(" O ", end=' ')
```

```
print()
```

```
board= [ [0,0,0],  
          [0,0,0],  
          [0,0,0] ]
```

```
menu()
```

Tuplas

O que são ? Para que servem ?

Uma característica de uma lista em Python, é que ela é mutável. Ou seja, podemos adicionar elemento, retirar, reordenar, mudar o valor de um item etc.

Tuplas também são sequências, mas são imutáveis. Logo, uma vez criada e declarada, você não pode fazer mais nenhuma alteração nela.

Um exemplo de sequência imutável, é a string. Depois que você declara ela, não pode mais alterar um caractere sequer. Até pode obter uma lista a partir dela, sub-strings etc, mas nunca vai poder alterar o valor delas na memória.

- **Como Declarar uma Tupla**

Nas listas, os valores vinham entre colchetes: []

Nas tuplas, os valores vem entre parêntesis: ()

Por exemplo:

- `aluno = ('Joaozinho', 10, 9)`

Pronto, **aluno** é uma tupla que possui uma string e dois inteiros dentro.

Caso queria criar uma tupla com apenas um elemento, deixe uma vírgula no final, para o Python saber que quer declarar uma tupla e não apenas algo entre parêntesis:

- `nota = (9.0,)`

- **Tupla - Quando usar?**

Você deve usar uma tupla sempre que tiver valores, items, que nunca serão mudados, serão constantes.

Por exemplo, se você for criar um programa que use coordenadas geográficas, cada ponto da terra pode ser representado por uma tupla, e este valor jamais deverá ser alterado.

Os endereços de memória, por exemplo, também são fixos, não mudam. Seu endereço, pode ser representado por uma tupla.

Duas vantagens das tuplas em relação a listas: são mais rápidas e protegem seus scripts de sofrerem alterações. Se você armazenar sua senha em uma tupla, um hacker nunca vai conseguir mudar essa senha.

Se convencionamos que listas são para item homogêneos (tudo inteiro, tudo string etc).

Já as tuplas são mais usadas para dados heterogêneos, ou seja, itens de string e número tudo junto na mesma tupla (como o nome e as notas de um aluno).

- **Transformar Tupla em Lista e vice-versa**

Muitas vezes vai ser necessário pegar os valores que estão em uma tupla e passar para uma lista, para aí podermos alterar.

Fazemos isso com a função **list()** que recebe uma tupla e transforma em lista.

Já se você quiser transformar uma lista em uma tupla, use a função **tuple()** que recebe uma lista e retorna uma tupla.

Dicionários

O que é ?

Esse tipo de dado é, sem dúvida, o mais poderoso e flexível, dentre as sequências.

Nas listas e tuplas, tínhamos uma coisa em comum: os itens eram ordenados, começavam no índice 0, e iam de um em um. Ou seja, são tipos de dados ordenados.

Já os dicionários são do tipo desordenados, pois não são numerados por índice.

Ao invés de índice, temos *chaves*.

E que chaves são essas? O que você quiser.

As chaves são *índices* também, mas pode ser qualquer número que você quiser ou uma string, por exemplo.

Eles não estão armazenados de uma maneira lógica e sequencial, estão propositalmente 'espalhados'

- **Dicionário - Como declarar e Usar**

Vamos definir um dicionário chamado **meuCachorro**.

Se nas listas usamos colchetes [], nas tuplas usamos parêntesis (), nos dicionários iremos usar chaves { }, veja:

```
meuCachorro={ 'raca': 'bulldog', 'idade' : 3, 'nome' : 'Florisbull'}
```

As chaves são: 'raca', 'idade' e 'nome'.

Elas tem os valores: 'bulldog', 3 e 'Florisbull', respectivamente.

Dicionário de uma maneira mais organizada:

```
meuCachorro={ 'raca': 'bulldog',  
               'idade' : 3,  
               'nome' : 'Florisbull'}
```

Assim, a raça do cachorro está armazenada em: **meuCachorro['raca']**

Já a idade dele está armazenada em: **meuCachorro['idade']**

E o nome em: **meuCachorro['nome']**

Ou seja, os itens de um dicionário sempre vem aos pares: chave e valor, separados por :

- **Exemplo de uso de Dicionário em Python**

No código abaixo, criamos um dicionário de nome **notas**, onde temos:

1. três chaves: joao, maria, zezinho
2. três valores: 9, 10 e 4.

Ou seja, as chaves são os nomes e os valores, as notas.

A seguir, damos três prints para mostrar os nomes e notas:

```
notas={'joao' : 9,  
      'maria' : 10,  
      'zezinho': 4}  
  
print("Nota de João: ",notas['joao'])  
print("Nota de Maria: ",notas['maria'])  
print("Nota de José: ",notas['zezinho'])
```

Nota a maneira como declaramos e inicializamos o dicionário, cada linha um par chave-valor.

Porém, isso não é obrigatório, poderíamos fazer assim:

```
notas={'joao':9,'maria':10,'zezinho':4}
```

Mas assim é mais organizado e bonitinho:

```
notas={'joao' : 9,  
      'maria' : 10,  
      'zezinho': 4}
```

Não concorda?

- **Exercício de Dicionário**

No dicionário abaixo, temos os dados de acesso de 3 usuários, cada um com seu login e senha, onde o login é a chave e a senha o valor.

Faça um script que peça ao usuário seu login e senha, se tiver certo envie uma mensagem de acesso autorizado, se fornecer a senha errada, informe o erro.

```
loginSenha={'joao' : 'rush', 'maria' : 'yes', 'zezinho': 'genesis'}
```

- Código Python:

Usaremos a variável *login* para armazenar o valor de uma chave do dicionário (pode ser joao, maria ou zezinho), em seguida vamos ver se o valor contigo em *loginSenha[login]* é o mesmo da senha que o usuário digitou.

```
loginSenha={'joao' : 'rush',  
            'maria' : 'yes',  
            'zezinho': 'genesis'}  
  
login=input("Qual seu login: ")  
senha=input("Senha: ")  
  
if loginSenha[login] == senha:  
    print("Acesso autorizado...")  
else:  
    print("Senha errada")
```

Note que se digitar 'joao' para login e digitar outra senha, como 'yes', vai dar erro.

Cada chave tem apenas um valor associado, são os **pares chave-valor**.

- **Dicionário** - Para que serve

Assim como as listas, os dicionários são mutáveis, ou seja, podemos inserir, retirar e mudar seus itens, tanto as chaves como os valores.

Não existe ordem, começo ou fim em um dicionário.

Por isso dizemos que são usados para *mapeamento*, pois mapeiam sua chave com seu valor.

Ou seja, acessamos os itens de um dicionário através de suas chaves.

É como se o **valor** só fosse aberto por uma determinada **chave**, como um cofre.

Como Exibir Dicionários

No tutorial anterior, aprendemos: [O que é um Dicionário, como funciona e para que serve.](#)

Vamos usar o exemplo dos alunos e das notas, onde temos um dicionário com três pares chave-valor, com nomes e notas dos alunos:

```
notas={'joao' : 9,  
      'maria' : 10,  
      'zezinho': 4}
```

```
print("Nota de João: ",notas['joao'])  
print("Nota de Maria: ",notas['maria'])  
print("Nota de José: ",notas['zezinho'])
```

Para exibir todos os itens de um dicionário em Python, podemos usar três **métodos** (que são funções que já vem junto com o dicionário):

1. Método items(): exibe todos os itens, ou seja, os pares chave-valor
2. Método keys(): exibe todas as chaves de um dicionário
3. Método values(): exibe todos os valores de um dicionário

- **Como exibir os itens: Método items()**

Se o nome de nosso dicionário é **notas**, para usar o método *items*, basta fazermos:

- notas.items()

Colocamos esse código dentro de um print.

- Código Python:

```
notas={'joao' : 9,  
      'maria' : 10,  
      'zezinho': 4}
```

```
print( notas.items() )
```

- Resultado:

```
===== RESTART: /home/user/dicionarios.py :  
dict_items([('joao', 9), ('maria', 10), ('zezinho', 4)])  
>>> [
```

Ou seja, apareceu:

```
dict_items([('joao', 9), ('maria', 10), ('zezinho', 4)])
```

- **Com exibir chaves de um dicionário: Método `keys()`**

Para exibirmos as chaves de um dicionário cuja variável se chama **notas**, usamos o método *keys*:

```
notas.keys()
```

- Código Python:

```
notas={'joao' : 9,  
      'maria' : 10,  
      'zezinho': 4}  
  
print("Exibindo chaves:")  
print( notas.keys() )
```

- Resultado:

```
===== RESTART: /home/user  
Exibindo chaves:  
dict_keys(['joao', 'maria', 'zezinho'])  
>>>
```

Apareceu:

```
dict_keys(['joao', 'maria', 'zezinho'])
```

- **Como exibir valores de um dicionário: Método `values()`**

Por fim, para exibirmos os valores de um dicionário **notas**, basta usarmos o método *values*:

```
notas.values()
```

- Código Python:

```
notas={'joao' : 9,  
      'maria' : 10,  
      'zezinho': 4}  
  
print("Exibindo valores:")  
print( notas.values() )
```

- Resultado:

```
===== RESTART: /home/user/dicionarios.py
Exibindo valores:
dict_values([9, 10, 4])
>>> |
```

O resultado é:

`dict_values([9, 10, 4])`

- **Exibindo Corretamente um Dicionário**

Ok, aprendemos os métodos para exibir separadamente os itens, os valores e as chaves.

Mas va lá, ficou feio.

Negócio de `dict_values` e `dict_values`, está estranho.

O melhor seria aparecer algo do tipo:

```
===== |
João  tirou nota: 9
Maria  tirou nota: 10
José  tirou nota: 4
>>> |
```

O grande segredo está nas **chaves**.

Vamos pegar o método **keys()** e usar a lista que ela retorna num laço **for**, veja a maneira mais indicada para exibir de uma maneira mais formatada um dicionário:

```
notas={'João' : 9,
      'Maria' : 10,
      'José': 4}
```

```
for nome in notas.keys():
    print(nome, " tirou nota: ", notas[nome])
```

O que fizemos foi pegar cada chave, na variável **nome** e usar ela pra acessar os valores corretos do dicionário.

Métodos dos Dicionários em Python

Uma das maiores vantagens do Python é já ter muita coisa 'embutida'.

Ou seja, já tem muita coisa pronta, feita, apenas usar o método, função, módulo ou biblioteca específica. Assim, você evita de ficar 'inventando a roda'.

Agora, vamos aprender a usar dois métodos, que são funções prontas para se trabalhar com dicionários.

Vamos usar o código do tutorial anterior:

- Como exibir items de um dicionário

Que é, basicamente, o dicionário:

```
notas={'João' : 9,  
      'Maria' : 10,  
      'José': 4}
```

Nos próximos tutoriais também vamos aprender a usar novos métodos.

- **Como verificar se uma chave existe:**
Método `get()`

Um dicionário não pode ter duas chaves iguais.
Já um valor pode se repetir várias vezes.

No dicionário anterior, podemos ter várias notas iguais, mas somente um João, uma Maria e um José, ok?

Experimente rodar o seguinte código:

```
notas={'João' : 9,  
      'Maria' : 10,  
      'José': 4}  
print("Nota do Neil Peart: ", notas['Neil Peart'])
```


Vai dar erro, pois não tem o aluno 'Neil Peart', vai aparecer uma mensagem de erro (*KeyError*).

Seu programa dá um *crash*, para de funcionar, se tentar acessar um item que não existem.

Geralmente, antes de trabalhar com um dicionário, especificamente quando queremos adicionar mais itens, é necessário verificar se uma chave já existe.

Para isso, vamos usar o método **get()**,

Esse método recebe dois parâmetros: `get(key, value)`

- `key` - É a chave que você quer testar, pra saber se existe
- `value` - Caso essa chave não existe, ele retorna esse valor ou *None* (caso não forneça nada, é opcional esse *value*)

O script abaixo testa se 'Peart' é um aluno ou não na lista:

```
notas={'João' : 9,  
      'Maria' : 10,  
      'José': 4}
```

```
if notas.get('Peart') == None:  
    print("Nao existe aluno Peart")  
else:  
    print("Aluno existente")
```

- **Definir o valor de uma chave:**
Método `setdefault()`

Note o que o método **get()** faz no dicionário quando a *chave* fornecida não existe: **nada**.

Apenas retorna para quem chamou o método o valor *key* de **get(key, value)**, uma espécie de resposta.

Seria mais útil se caso não existisse essa *key* ele criasse um item com essa chave e atribuísse a ele o valor *value*, não acha?

Se é óbvio, pode ter certeza que tem no Python.
Fazemos isso com o método **setdefault(key, value)**

O script abaixo testa se tem o aluno 'Peart', se tiver ele cria um item com esta *key* e dá o valor 8 para ele. Em seguida, imprimimos o dicionário para você ver como esse aluno foi adicionado, vejam:

```
notas={'João' : 9,  
      'Maria' : 10,  
      'José': 4}  
  
notas.setdefault('Peart', 8)  
print(notas)
```

Caso a chave já exista, o método *setdefault* não faz **nada** na chave existente, não altera nada nela, ok?

Dicionário: Como adicionar, Alterar e Retirar um Item em Python

- **Adicionar item em um Dicionário**

Vamos pegar o nosso bom e velho dicionário **notas**:

```
notas={'João' : 9,  
      'Maria' : 10,  
      'José' : 4}
```

Vamos supor que chegou um novo aluno, o *Peart*, que já chegou tirando nota 10 também.

No tutorial anterior ([método get e setdefault - Como usar](#)), aprendemos como inserir um valor através do método **setdefault**.

Você também pode inserir um item em um dicionário **dict** com chave **key** e valor **value** simplesmente fazendo a operação direta:

- dict[key] = value

Por exemplo, vamos adicionar o aluno *Peart*:

```
notas={'João' : 9,  
      'Maria' : 10,  
      'José' : 4}
```

```
notas['Peart'] = 10  
print(notas)
```

Simples, não é verdade?

Mas, cuidado, se esta chave já existir, ele vai substituir o valor anterior.

Por isso, é bom só fazer isso se **não** existir essa nova chave.

Podemos checar isso com o **get**.

O script abaixo pergunta um nome e uma nota.

Se este nome já existir, o método **notas.get(nome)** vai retornar algum valor do dicionário, e cai no **IF** dizendo que o nome já existe.

Se a chave não existir, retorna *None* por padrão, o método `get()`, então cai no ELSE, onde fazemos a atribuição e inserimos um novo item no dicionário, veja:

```
notas={'João' : 9,  
      'Maria' : 10,  
      'José' : 4}
```

```
nome = input("Digite o nome do aluno: ")  
nota = float(input("Nota dele: "))
```

```
if notas.get(nome):  
    print("Ja existe o aluno ",nome)  
else:  
    notas[nome] = nota  
print(notas)
```

- **Como Alterar um Item: *in***

Agora, vamos aprender como mudar o valor de um determinado item.

A primeira coisa que devemos fazer é confirmar se realmente aquele item existe.

E podemos fazer isso com a instrução **in**.

Se fizermos **key in notas.keys()**

Ele vai avaliar se a chave *key* está na lista de valores de *notas.keys()*

Se sim, retorna *True* e cai no IF para podermos alterar o valor.

Se não tiver, retorna *False* e no ELSE dizemos que o aluno não existe, logo não dá pra alterar o valor dele.

Então, nosso script pra alterar alguma nota, ficaria assim:

```
notas={'João' : 9,  
      'Maria' : 10,  
      'José' : 4 }
```

```
nome = input("Aluno a mudar a nota: ")  
nota = float(input("Nova nota : "))
```

```
if nome in notas.keys():
```

```
    notas[nome] = nota
else:
    print("Não existe esse aluno")
print(notas)
```

- **Como excluir um item: `pop()`**

Para excluirmos um determinado item, basta usarmos o método **`pop()`** que vai receber a *key* de deletar o item cuja chave é aquela.

Por exemplo, para retirarmos o 'José' do dicionário:

```
notas={'João' : 9,
      'Maria' : 10,
      'José' : 4 }
```

```
print(notas)
notas.pop('José')
print(notas)
```

O resultado é:

```
===== RESTART: /home/u
{'João': 9, 'Maria': 10, 'José': 4}
{'João': 9, 'Maria': 10}
>>> |
```

Exercício de Dicionários em Python

Uma escola te contratou para fazer um *software* em Python.

Eles querem que você crie um *script* que exiba o seguinte menu:

0. Sair
1. Exibir lista de alunos com suas notas (cada aluno tem uma nota)
2. Inserir aluno e nota
3. Alterar a nota de um aluno
4. Consultar nota de um aluno específico
5. Apagar um aluno da lista
6. Dar a média da turma

Onde a professora que vai fornecer o nome e nota dos alunos. Quantos ela quiser. Quantas vezes quiser.

Implemente esse script usando um dicionário.

Resolução:

Primeiro, criamos um dicionário de nome **alunos** e depois chamamos a função que exibe o menu.

- menu()

Bem, vamos lá.

Primeiro, temos que criar a lógica que vai ficar exibindo esse menu o tempo inteiro.

Ele é exibido enquanto o valor da variável **continuar** for diferente de 0.

Se for, encerra o menu() e acaba o laço e encerra o programa.

Se for outro valor diferente, vai cair em vários testes condicionais para saber que função correta deve ser chamada.

- exibir()

Nessa função, vamos simplesmente exibir o dicionário, de maneira bonita e padronizada, os nomes e notas dos alunos.

Para isso, usamos uma variável temporária de nome *nome*, que vai receber cada uma das chaves (*keys*), do dicionário de alunos.

Então, imprime o nome e a nota de cada aluno.

- inserir()

Antes de inserir, usamos o método **get()** para saber se já existe uma *key* com aquele nome que estamos querendo inserir.

Se tiver o aluno, o método retorna algum valor, cai no IF e avisamos que não pode inserir.

Se não estiver o aluno, cai no ELSE e criamos um novo item no nosso dicionário.

- alterar()

Antes de alterar um determinado item, temos que nos certificar que ele existe no dicionário.

Para isso, usamos o operador **in**.

Se ele existir na lista de *keys*, faz a alteração.

Se não não faz nada, só exibe uma mensagem avisando isso.

- consultar()

Idem a lógica anterior.

Se existir o item, exibe.

Senão, avisa.

- apagar()

Idem.

Primeiro, verificamos se o item existe, consultando sua chave.

Se tiver, usamos o método **pop()** para retirar o item da lista.

- media()

Pegamos uma variável e inicializamos com valor 0.

Depois, somamos todos os valores nessa variável e dividimos pelo tamanho do dicionário, através da função **len()**.

- **Código Python**

```
def menu():
    continuar=1

    while continuar:
        continuar = int(
            input("0. Sair\n"+
                "1. Exibir lista de alunos com suas notas (cada aluno tem uma
nota)\n"+
                "2. Inserir aluno e nota\n"+
                "3. Alterar a nota de um aluno\n"+
                "4. Consultar nota de um aluno específico\n"+
                "5. Apagar um aluno da lista\n"+
                "6. Dar a média da turma\n"))
        if continuar==1:
            exibir()
        elif continuar == 2:
            inserir()
        elif continuar == 3:
```

```
    alterar()
elif continuar == 4:
    consultar()
elif continuar == 5:
    apagar()
elif continuar == 6:
    media()
elif continuar == 0:
    print("Encerrando programa")
else:
    print("Opção inválida")
```

```
def exibir():
    for nome in alunos.keys():
        print("Nome: ", nome, " - Nota: ", alunos[nome])
```

```
def inserir():
    nome = input("Digite o nome do aluno: ")
    nota = float(input("Nota dele: "))
```

```
    if alunos.get(nome):
        print("Ja existe o aluno ", nome)
    else:
        alunos[nome] = nota
```

```
def alterar():
    nome = input("Aluno a mudar a nota: ")
    nota = float(input("Nova nota : "))
```

```
    if nome in alunos.keys():
        alunos[nome] = nota
    else:
        print("Não existe esse aluno")
```

```
def consultar():
    nome = input("Digite o nome do aluno: ")

    if alunos.get(nome):
        print("Nota de ", nome, ": ", alunos[nome])
    else:
        print("Nao existe tal aluno")
```

```
def apagar():
    nome = input("Apagar que aluno: ")
```



```
if alunos.get(nome):  
    alunos.pop(nome)  
else:  
    print("Não existe o aluno ", nome)
```

```
def media():  
    soma = 0  
    for count in alunos.values():  
        soma += count  
    print("Média dos alunos: %.2f" % (soma / len(alunos) ))
```

```
alunos = {}  
menu()
```

Como Mudar a Key (Chave) de um Dicionário

No tutorial anterior, aprendemos a [inserir, alterar e excluir um item](#).

Porém, alteramos o valor de uma determinada chave.

Por exemplo, no dicionário:

```
notas={'João' : 9,  
      'Maria' : 10,  
      'José' : 4 }
```

Mudamos a nota de 'Maria' de 10 para 9 fazendo:

- `notas[Maria] = 9`

Mas e se eu quiser mudar 'Maria' para 'Marya', pois eu digitei errado o nome dela?

Existem duas maneiras.

- **Maneira 1 de mudar a key**

Primeiro, definimos uma nova *key*, com o valor da antiga.

Depois, apagamos a antiga:

```
notas={'João' : 9,  
      'Maria' : 10,  
      'José' : 4 }
```

```
print(notas)
```

```
notas['Marya'] = notas['Maria']  
del notas['Maria']
```

```
print(notas)
```

Teste o código anterior!

- **Maneira 2 de alterar a key**

Podemos fazer isso também usando o **método pop()**, pois primeiro ele retorna a *key* que recebe, e automaticamente descarta ela depois:

```
notas={'João' : 9,  
      'Maria' : 10,  
      'José' : 4 }
```

```
print(notas)  
notas['Marya'] = notas.pop('Maria')  
print(notas)
```

Bem mais simples, não?

Em ambos casos temos o seguinte resultado na tela:

```
===== RESTART: /home/u  
{ 'João': 9, 'Maria': 10, 'José': 4}  
{ 'João': 9, 'José': 4, 'Marya': 10}  
>>>
```

Ou seja, no fim das contas, não alteramos de fato a *key*, pois isso não é permitido.

Apenas copiamos o seu *value* anterior e colocamos ele em uma nova chave, com novo nome.

Como Copiar um Dicionário ou Lista

- **Cópia rasa (*shallow copy*)**

Você foi contratado por uma grande empresa multinacional para programar e gerenciar o sistema dela.

Em um dicionário, você salvou os dados de login e senha de alguns funcionários, com nome **login**:

```
login={'joao' : 'senha',  
      'maria' : '123456',  
      'zezinho': 'restart'}
```

Por exemplo, o login 'joao' tem senha 'senha'. Já a senha de 'maria' é '123456' e a senha do 'zezinho' é 'restart'.

Como você é um bom programador, afinal estudou pelo Curso Python Progressivo, decidiu criar um *backup* e salvar esses dados.

Criou a variável **backup** e salvou uma cópia do dicionário **login**.
`backup = login`

Pronto, agora o dicionário **backup** tem uma cópia dos dados de **login** da empresa.

Porém, a galera descobriu que a senha do zezinho é 'restart' e ficou zoando ele.

Ele pediu pra você mudar a senha dele para 'ironmaiden'.

Você fez o seguinte:

```
login['zezinho'] = 'ironmaiden'
```

Pronto, senha do zezinho foi mudada.

Agora vamos imprimir o dicionário **login** (com a alteração) e a cópia antiga, o **backup** (antes da cópia):

Olha só!

Você alterou o dicionário **login**, mas o dicionário **backup** também foi alterado!

Por que isso aconteceu ?

- **Referência e Cópia rasa**

Quando definimos a variável **login** e criamos nosso dicionário, reservamos um espaço na memória do computador.

Quando fazemos: **backup = login**

Estamos criando a variável *backup*, porém ela não é um dicionário, ela vai simplesmente armazenar uma *referência* para a variável **login** (essa sim é um espaço na memória com um dicionário lá).

Ou seja, a variável *backup* tem um endereço de memória (referência), que aponta pro local de memória de *login*.

Por isso, quando alteramos **login**, automaticamente alteramos **backup**, pois **backup** aponta para aquele endereço de memória de **login**, que alteramos!!!

Ou seja, não fizemos uma **cópia** de verdade, fizemos uma cópia rasa. Vamos aprender como fazer uma cópia de verdade.

- **Cópia profunda (*deep copy*): `copy.deepcopy()`**

Primeiro, importamos o módulo **copy**

```
import copy
```

E pronto! Aí sim criamos uma cópia de verdade.

Agora, a variável *backup* aponta para um endereço de memória diferente do endereço de memória onde está a variável **login**.

Nesse novo endereço, tem um outro dicionário diferente, um novo. É uma cópia profunda (do inglês, *deep*).

```
import copy
```

```
login={'joao' : 'senha',  
      'maria' : '123456',  
      'zezinho': 'restart'}
```

```
backup = copy.deepcopy(login)
```

```
print(backup)
```

```
login['zezinho'] = 'ironmaiden'
```

```
print(login)
```

```
print(backup)
```

Ou seja, cuidado ao passar listas e dicionários por referência!

As vezes queremos passar a referência e as vezes uma cópia (profunda), cuidado, moçada!

Arquivos

Até o momento, todas nossas informações ficaram armazenadas em variáveis.

Esses dados ficam salvos durante a execução dos scripts Python. Porém, quando fechamos, essas variáveis 'somem', pois seus endereços de memórias antes alocados para elas, ficam livres novamente.

Nesta seção, iremos te ensinar como fazer para salvar, escrever e ler dados que ficam armazenados em arquivos de seu computador.

Assim, podemos salvar informações em arquivos, fechar o script, depois abrir novamente o programa e obter de volta esses dados, escrever mais, apagar etc.

São ditas informações persistentes, pois elas persistem, continuam a existir mesmo que você o programa ou seu computador, as informações ficarão armazenadas em seu HD.

Como Abrir e Ler um arquivo: `open()` e `read()`

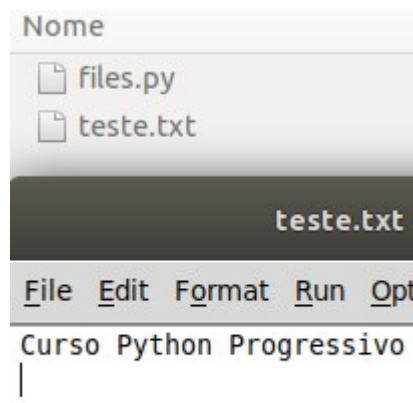
- **Com Abrir um Arquivo em Python: `open()`**

Antes de abrir um arquivo, vamos criar ele.

Na pasta em que vai ficar armazenado seu *script*, crie um arquivo chamado: **teste.txt**

Dentro dele escreva algo, como "Curso Python Progressivo".

Veja como ficam os arquivos (mesma pasta) e arquivo de texto aberto com uma *string*:



Para abrirmos o arquivo **teste.txt** vamos usar a função **`open()`**.

Essa função recebe uma *string*, com o endereço do arquivo que você deseja abrir.

Como nosso arquivo está no mesmo diretório do script, podemos escrever só seu nome: 'teste.txt' (não se esqueça das aspas, precisa ser uma string).

E prontinho, essa função retorna objeto do tipo *File* (não se preocupe, em breve vamos aprender melhor o que é um objeto). Para não perdermos esse objeto, vamos associar ele a uma variável, a **meuArquivo**

Então, para abrir um arquivo de nome *teste.txt*, usamos o comando:

```
meuArquivo = open('teste.txt')
```

Experimente imprimir agora a variável **meuArquivo**:


```
meuArquivo = open('teste.txt')
print(meuArquivo)
```

Vai aparecer a seguinte informação:

```
<_io.TextIOWrapper name='teste.txt' mode='r' encoding='UTF-8'>
>>>
```

Ou seja, um punhado de informações sobre o arquivo *teste.txt*, como seu nome, modo de abertura (vamos explicar mais na frente) e codificação dele.

- **Como Ler um Arquivo: `read()`**

Ok, abrimos um arquivo e já temos um 'meio' de se comunicar e trabalhar com o arquivo, através da variável **meuArquivo**.

Mas queremos acessar é o conteúdo dele, ver o que tem dentro.

Para isso, basta usarmos o **método `read()`**, que tem nos objetos do tipo File.

Para imprimir o conteúdo de um arquivo, é só fazer:

- `meuArquivo.read()`

Vejamos um código onde abrimos um arquivo e depois imprimimos seu conteúdo:

```
meuArquivo = open('teste.txt')
print(meuArquivo.read())
```

E prontinho! Aparece no *shell* do Python, quando você rodar o script, o conteúdo do arquivo txt.

Bacana, não ?

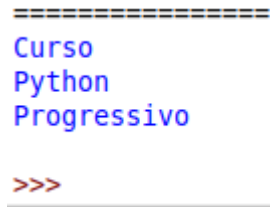
Agora, em vez de deixar escrito no arquivo o texto 'Curso Python Progressivo', escreva:

Curso

Python

Progressivo

E rode novamente o programa, o resultado vai ser:



Ou seja, ele copiou idêntico ao arquivo, inclusive as quebras de linha. Mesmo que o arquivo tenha 1 milhão de linhas de texto, o método **read()** vai retornar apenas uma string, com o texto inteiro do arquivo, ok?

E repetindo:

- meuArquivo - é um objeto do tipo File
- meuArquivo.read() - é uma string

São coisas diferentes!

Como Ler Linha por Linha em Arquivo em Python: `readlines()`

- **Ler Linha por Linha: Método `readlines()`**

No tutorial anterior de nosso curso, falando como [abrir e ler um arquivo](#) em Python.

Para isso, usamos a função `open()` e o método `read()` do objeto do tipo *File*.

Porém, quando lemos usando o `read()`, ele pega todo o conteúdo do arquivo de texto, e coloca em apenas uma única string, tudo junto.

Porém, geralmente, não é isso que desejamos, e sim pegar 'pedaços' menores.

Por exemplo, crie o arquivo de texto com o nome dos funcionários de sua empresa:

```
Geddy Lee  
Neil Peart  
Alex Lifeson
```

Salve como **funcionarios.txt** e deixe na mesma pasta (diretório) do seu script Python.

Aprendemos que para abrir, ler e printar o conteúdo do arquivo, fazemos:

```
meuArquivo = open('funcionarios.txt')  
nomes = meuArquivo.read()  
print(nomes)
```

Ou seja, na variável *nomes* tem uma string gigante, com o nome dos três funcionários.

Existe um método, chamado **`readlines()`**, que também faz parte do objeto do tipo *File*.

Quando fazemos isso, ele lê o arquivo, porém ao invés de retornar uma *string* ele retorna uma LISTA DE STRINGS ! Ele vai pegar linha por linha, e cada linha vai ficar em uma posição da lista.

Veja como fica nosso código usando **`readlines()`**:

```
meuArquivo = open('funcionarios.txt')
nomes = meuArquivo.readlines()
print(nomes)
```

O resultado do print, no shell, é:

```
['Geddy Lee\n', 'Neil Peart\n', 'Alex Lifeson\n']
```

Ou seja, uma lista, com três strings dentro!

Para a *readlines()* cada linha é uma string que termina no caractere '\n'.

Seu arquivo de texto pode ter 1 milhão de linhas.

Se usar a *readlines()*, ela vai criar uma lista de 1 milhão de elementos.

- **Exercício de Arquivos em Python**

Copie toda a letra da música 2112, da banda Rush (melhor banda do universo):

<https://www.letras.mus.br/rush/80867/>

E salve em um arquivo de texto **letra.txt**

Depois, crie um script de Python que exibe essa letra e o número de linhas dessa letra.

Nosso código Python ficou:

```
meuArquivo = open('letra.txt')
nomes = meuArquivo.readlines()
```

```
for nome in nomes:
    print(nome)
```

```
print("Número de linhas na letra: ", len(nomes))
```

Note que para saber o número de linhas, basta usarmos a função **len()** com o valor da lista dentro dessa função, ela vai retornar o número de strings na lista, que é o mesmo número de linhas da letra.

Aqui, deu que tem 144. Aí também?

Como Escrever em um Arquivo: `write()` e Modos de Abertura

- Como Escrever em um Arquivo: Método `write()`

A palavra *print* em inglês significa imprimir.

Quando usamos a função *print* é porque estamos imprimindo, escrevendo na tela do shell e na tela de seu computador, é uma maneira de escrever.

Outra maneira de *printar* ou escrever é fazendo isso em arquivos. Ou seja, escrevemos via script Python dentro de um arquivo.

Mas, antes, precisamos ver o que é modo de abertura.

- **Modos de Abertura**

Quando falamos em [abrir arquivos via função `open\(\)`](#) fazemos somente:
`open(caminho)`

Onde *caminho* é uma string com o endereço do arquivo que queremos abrir. Porém, isso é o mesmo que fazer:

- `open(caminho, 'r')`

Esse *r* é de *read*, ou seja, leitura. É um modo de abertura, significa que estamos abrindo um arquivo para leitura, ou seja, modo de abertura *read*.

Nesse modo, podemos apenas ler os dados de um arquivo, não podemos alterar nada.

- **Escrevendo em Arquivos: `write()` e `'w'`**

Antes de escrevermos em um arquivo, precisamos abrir ele no modo *escrita*. Para isso, fazemos:

- `open(caminho, 'w')`

Em seguida, para escrevermos uma *string* em um arquivo de nome ***file.txt*** usamos o método **`write()`**, de objetos de tipo *File*.

Por fim, temos sempre que **fechar** o arquivo, ao término do script, através do método **close()**.

O seguinte código Python escreve a string "Curso Python Progressivo" no arquivo **file.txt**:

```
meuArquivo = open('file.txt', 'w')
meuArquivo.write("Curso Python Progressivo")
meuArquivo.close()
```

Agora vá lá na pasta que está o seu script Python e veja que apareceu um arquivo de nome **file.txt**

Abra ele e veja que vai estar escrito lá dentro:

"Curso Python Progressivo".

Prontinho! Você acabou de escrever, printar uma string dentro de um arquivo em seu computador.

Detalhe para uma coisa: se você abrir e escrever em um arquivo que não existe, ele cria esse arquivo.

- **Adicionando Informações em um Arquivo: Modo *append*, 'a'**

Agora temos um arquivo de nome **file.txt** e dentro dele tem escrito "Curso Python Progressivo".

Vamos escrever outra frase nele:

"O melhor curso de Python"

Repetimos o código:

```
meuArquivo = open('file.txt', 'w')
meuArquivo.write("O melhor curso de Python")
meuArquivo.close()
```

Agora vá lá e leia o que está escrito:

"O melhor curso de Python"

Epa!!!! Alto lá!!!

Cadê a string "Curso Python Progressivo" ? Queríamos adicionar uma string, e não substituir o que tinha lá.

Pois é, ao usar o modo de abertura *write* 'w', ele sobrescreve o arquivo, caso ele já exista.

Ou seja, apaga e cria um novo, com essa nova informação.

Para ADICIONAR, vamos usar o modo de adição (*append mode*), simbolizado por 'a' quando formos abrir nosso arquivo.

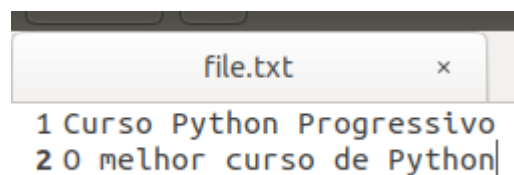
Repita o código lá de cima, com:

```
meuArquivo = open('file.txt', 'w')
meuArquivo.write("Curso Python Progressivo\n")
meuArquivo.close()
```

Agora, modifique seu código para:

```
meuArquivo = open('file.txt', 'a')
meuArquivo.write("O melhor curso de Python")
meuArquivo.close()
```

E abra o arquivo de texto:



Voilà! Adicionamos mais uma string, sem apagar o que tinha lá antes!!!
Tudo isso porque usamos o modo de abertura 'a'.

Como Retirar a Quebra de Linha (caractere \n) em Python

- **O Caractere de Quebra de Linha em Arquivos: \n**

Escreva em um artigo de texto, de nome **bandas.txt** o nome de algumas bandas favoritas que você tem.

As minhas bandas favoritas são:

Rush

Iron Maiden

Led Zeppelin

O arquivo deve conter só isso.

3 bandas, 3 linhas de informação.

Agora vamos criar um *script* em Python que exibe essas minhas três bandas favoritas.

Como já aprendemos em [como ler um arquivo linha por linha com a readlines\(\)](#):

```
meuArquivo = open('bandas.txt', 'r')
bandas = meuArquivo.readlines()
for banda in bandas:
    print(banda)
```

Prontinho, a variável **banda** assume o valor de cada linha do arquivo de texto **bandas** e imprime as bandas, o resultado é:

```
=====
Rush

Iron Maiden

Led Zeppelin

>>> |
```

Mas...peraí...tem uma coisa estranha, me incomodando:

Tem uma linha em branco ali, entre cada banda.

Que troço chato, que troço feio. Como se retira isso?

Aliás, por que aparece essa linha em branco a mais ?

A explicação é simples:

Ao digitar "Rush", você não digita "Rush".

Você digita "Rush\n", pois você dá um **enter** ali no final, pra quebrar a linha, pra ir pra linha de baixo.

Então, quando a variável *banda* vai pegar a string, ela não pega:
"Rush"

Ela pega:

"Rush\n"

Por isso aparece sempre uma linha nova, em branco, quando damos print.
Temos que remover essa maldita *new line* \n

Quando você vai jogar algum jogo, aparece:

"Fase: 5"

"Poder: 10"

"Defesa: 6"

E não:

"Fase: 5"

"Poder: 10"

"Defesa: 6"

Vamos lá! Vamo aprender a fazer o negócio direito!

- **Como Remover o Caractere de New Line: `rstrip('\n')`**

Como explicamos, ao lermos (`read()` ou `readlines()` ou `readline()`) um arquivo, vamos pegar ele na forma de *string*, ou seja, é um texto.

Vamos usar um método chamado `rstrip()`, ele serve pra 'estripar' algum caractere, ou seja, retirar, dar um *strip* nele, e varrer da string.

No caso, queremos retirar o caractere de nova linha, então fazemos: **rstrip('\n')**

Ele é um método de objetos do tipo *String*.

No caso, a variável que vai armazenar as *strings*, uma por uma, é a variável **banda**, então aplicamos nela o método:

banda.rstrip('\n')

Ela retorna a mesma *string* *banda*, mas sem os caracteres `\n`.

Veja como fica nosso código:

```
meuArquivo = open('bandas.txt', 'r')
bandas = meuArquivo.readlines()
```

```
for banda in bandas:
    banda = banda.rstrip('\n')
    print(banda)
```

E o resultado:

```
=====
Rush
Iron Maiden
Led Zeppelin
>>> |
```

Note que alteramos somente a variável temporária do *LAÇO FOR*, a **banda**. Para alterarmos cada elemento da **lista de strings** *bandas*:

```
meuArquivo = open('bandas.txt', 'r')
bandas = meuArquivo.readlines()
```

```
for index in range(len(bandas)):
    bandas[index] = bandas[index].rstrip('\n')
```

Agora, toda a nossa lista **bandas** não tem mais o maledito caractere `'\n'` ao final de cada string.

Como Processar Arquivos com Laço FOR (looping)

- **Arquivos e Laço FOR**

Até o momento, em nossa seção de [Arquivos em Python](#), trabalhamos com pequenos arquivos de textos, geralmente de duas ou três linhas, até para simplificar e agilizar nossos tutoriais de explicações.

Porém, o normal é termos arquivos grandes, com centenas ou milhares de linhas.

Quando você se tornar um programador Python profissional, vai ter planilhas documentos de texto, PDF e outros tipos de listas (como de funcionários e alunos), de milhares e milhares de linhas.

Então, vamos te mostrar como trabalhar com um arquivo usando o [laço FOR](#).

Primeiro, entre no seguinte link e copie toda a lista:

- [Lista de linguagens de programação](#)

Salve em um arquivo de texto de nome **linguagens.txt**

Vamos criar um script que pergunta ao usuário quantos nomes de linguagens de programação ele deseja ver.

Primeiro, criamos um objeto do tipo *FILE* de nome **arquivo**, para abrir `linguagens.txt`

Vamos pegar todo o conteúdo de texto e colocar numa lista de nome **linguas**, através do [método readlines\(\)](#)

Usamos a função **len()** para contar quantos elementos tem a lista *linguas*, cujo número de elementos é o número de linhas do arquivo de texto, logo, mostrando o total de linguagens de programação que tem no texto.

Perguntamos ao usuário quantas linguagens ele deseja ver o nome e armazenamos na variável inteira **num**.

Por fim, entramos no laço FOR.

Vamos percorrer um total de *num* elementos, onde imprimimos o número de vezes que o usuário escolheu, imprimindo os elementos da lista *linguas* (usamos o [método `rstrip\(\)`](#) para retirar o caractere de quebra de linha `\n`).

Veja como ficou nosso código:

```
arquivo = open('linguagens.txt', 'r')
linguas = arquivo.readlines()

print("Existem", len(linguas), "linguagens. Quantas deseja ver?")
num = int(input())

for count in range(num):
    print(str(count+1), ":", linguas[count].rstrip('\n'))

arquivo.close()
```

- **Detectando o final de um arquivo: *End of file*** "

Tem um probleminha no código anterior.
Experimente digitar 717.

Aguarde toooda a lista de linguagens de programação ser *printada* em sua tela e...erro:

```
716 : Z++
Traceback (most recent call last):
  File "/home/user/arquivos.py", line 9, in <module>
    print(str(count+1), ":", linguas[count].rstrip('\n'))
IndexError: list index out of range
>>> |
```

Deu erro de *list index out of range*, ou seja, você tentou acessar um índice da lista que não existe.

A lista tem 716 linguagens/linhas, e você tentou acessar mais que isso.

Poderíamos evitar esse erro fazendo uma coisa bem simples: **detectando o final do arquivo**.

Agora, ao invés de criar uma lista, vamos imprimir linha por linha do arquivo, através do método **`readline()`**, que lê uma linha por vez e retorna essa string, e armazena na variável **linha** que criamos.

Vamos usar o mesmo laço FOR pra imprimir o número de linguagens que o usuário quis ver.

Ao final, quando chega no fim do arquivo, o método `readline()` retorna uma string **vazia**.

Uma string vazia é simbolizada, simplesmente, por: ""

Ou seja, a cada iteração do *loop* verificamos o conteúdo de *linha*.

Se for diferente de string vazia, imprimimos mais uma linha.

Se for vazia, dizemos que chegou no fim do arquivo.

Nosso código Python ficou:

```
arquivo = open('linguagens.txt', 'r')
num = int(input("Numero de linguagens: "))
```

```
for linha in arquivo:
    linha = arquivo.readline()

    if linha != "":
        print(linha.rstrip("\n"))
    else:
        print("Fim do arquivo")
```

```
arquivo.close()
```

- **Lendo Arquivos com laço FOR**

Os exemplos anteriores foram mais para você aprender como pensa um programador, entender como as coisas funcionam por trás dos panos e te ajudar em lógica, ou seja, de **realmente** entender o que tá fazendo e o que tá acontecendo quando seu código roda.

Porém, existe uma maneira ainda mais simples de ler um arquivo de texto, sem usar **readlines()** nem a **readline()**, simplesmente imprimindo direto da variável do tipo *File*, e também sem precisar detectar o final de um arquivo. Sabe onde fizemos:

- for **variavel** in **range(valor)**:

Em vez de **range()**, basta usar o objeto do tipo *File* mesmo.
O Python subentende que seu *range* vai ser as linhas do arquivo.

Fica:

- **for** linha **in** arquivo:

Prontinho.

A cada iteração do *loop For* o valor de **linha** é uma linha do arquivo de texto **arquivo**, que definimos e abrimos anteriormente (com a *open()*).

Para exibir o número correto de linguagens que o usuário solicitou, vamos usar uma variável para fazer a contagem, a **count**. O IF imprime uma linha, já quando essa variável passa do valor digitado pelo usuário, cai no ELSE e ele dá um *break* no laço FOR, terminando a leitura do arquivo.

Veja como ficou nosso código Python:

```
arquivo = open('linguagens.txt', 'r')
num = int(input("Numero de linguagens: "))
count=0
```

```
for linha in arquivo:
    if count < num:
        print(linha.rstrip('\n'))
        count = count+1
    else:
        break
```

```
arquivo.close()
```

Módulo os : Caminhos, Endereços, Pastas e Diretórios

- **Acessando Outros Locais: *path***

Durante nossa seção de [Arquivos em Python](#), sempre que abrimos nossos arquivos, fornecemos apenas o nome do arquivo de texto, para a função **open()**

Por exemplo, para ler um arquivo cujo nome é *teste.txt*, fazemos:

```
•meuArquivo = open('teste.txt', 'r')
```

Ok, isso é certo e bonitinho se seu script Python estiver na mesma pasta ou diretório do arquivo de texto. Mas geralmente não é assim que se trabalha com arquivos.

As vezes nosso script está em:

C:\Python\scripts

E você quer hacker seu Windows lendo um arquivo:

C:\System32\teste.txt

E aí, como faz, José?

Simples, ao invoes de 'teste.txt', use o endereço completo (*path*) do arquivo:

```
•meuArquivo = open('C:\\System32\\teste.txt', 'r')
```

Note que usamos duas barras.

Tente imprimir no Python: **print('\')**

Vai dar erro.

O motivo é que, para imprimir uma barra \ precisamos fazer \\ (lembra do caractere '\n', pois é, o caractere \ é representado por '\\').

Exercício de Arquivos

Crie um script que crie um arquivo de nome 'virus.txt' em sua área de trabalho, e ao abrir, esteja escrito o texto: "Você foi hackeado. Curso Python Progressivo"

- Endereços nos Windows, Linux e Mac

Temos um pequeno 'problema' em relação aos endereços de arquivos e pastas, no que se refere a Linux/Mac e sistemas Windows.

Uma pasta do Windows: C:\Windows\Jose\Python

Um diretório no Linux : /home/user/Maria/Python

Ou seja, as barras são invertidas!

Se tentar acessar um endereço que não existe (como um C: no Linux ou /home no Windows) vai dar erro sim em seus scripts!

Assim, como bons programadores (afinal estudamos pelo curso Python Progressivo), temos que nos preparar para os dois sistemas, pois o Python por si só já tem um conceito de portabilidade e multiplataforma muito fortes, e devemos sempre ter isso em mente: nossos scripts devem rodar em qualquer sistema, até em Android se possível.

Vamos ver um 'truque' pra nos ajudar com isso, é a função **os.path.join()** do módulo **os**, que recebe várias strings, com os nomes das pastas e retorna uma string do endereço correto no sistema operacional de cada pessoa.

Por exemplo, se digitar: **os.path.join('home', 'user', 'Maria')**

Se estiver no Windows, a função retorna a string: 'home\\user\\Maria'

Se estiver no Linux, retorna a string: 'home//user//Maria'

- **Caminhos Absolutos e Relativos**

Quando você instala um jogo, e quando você for criar seus *games* e *softwares*, a primeira coisa que vai pedir ao usuário, é o local no sistema dele onde vai instalar.

Se ele disser: C:\Jogos selecionando lá as janelinhas do explorer, você guarda numa variável:

path = 'C:\\Jogos' e pronto.

Vai criar a pasta de fases como? Assim: **os.path.join(path, 'level')**

Pronto, te retorna a string com o endereço absoluto completo (desde C:\ atéeee 'level')

E a pasta de personagens? Assim: **os.path.join(path, 'characters')**

E a pasta do personagem 'Geddy'? **os.path.join(path,'characters','Geddy')**
E assim vai indo.

Note que não importa o que está na variável **path**, foi o usuário que escreveu.

Ela pode ser desde nada, vazia, até C:\usuario\secreto\porn\virus\nao_mexa'

Não importa, ok?

É por isso que te pedem pra navegar e escolher a pasta que vai querer instalar, pra você fornecer ao programa ou jogo, uma string com o caminho onde vão ficar as pastas e arquivos do *soft*.

- **Pasta atual: `getcwd()`**

Rode o seguinte script em Python:

```
import os
print( os.getcwd() )
```

O resultado vai ser um endereço de sua máquina.

Qual? O endereço em que você está rodando seu script.

O comando **getcwd()**, do módulo **os** (de sistema operacional) retorna o diretório de trabalho atual (*current working directory*, *cwd*).

- **Como mudar de pasta: `chdir()`**

Já o **chdir** recebe uma string, com um endereço pra onde você deseja ir.

Escolha um local, como sua área de trabalho (não sei que sistema você usa, nem suas pastas internas).

Como eu rodo Linux, vou dar uma passeada na minha pasta onde guardo meus scripts de Python.

Primeiro vou pra lá com **chdir**, depois printo o endereço de lá com **getcwd**, vejam meu código:

```
import os
os.chdir('//home//user//Python')
print( os.getcwd() )
```

Como ficou o de vocês, pra ir até sua pasta de scripts?

- **Como criar uma nova pasta: `makedirs()`**

E que tal criar uma pasta, dentro do diretório atual?

Basta fornecer o endereço na formada de string, para a função **`makedirs()`**
Vamos criar uma pasta chamada 'arquivos', dentro da minha pasta de scripts Python, lá onde vou guardar meus scripts sobre arquivos, faço assim:

```
import os  
os.makedirs('//home//user//Python//arquivos')
```

Crie uma sua também.

Depois vá lá e veja sua pasta criada.

Se não tivesse, por exemplo, a pasta 'Python', ele ia criar a 'Python' e depois a 'arquivos'.

Ele vai criando tudo, até criar a pasta existente que você deseja.

- **Basename e dirname**

Se tiver uma string com endereço de um arquivo:

```
path = 'C:\\Estudo\\Python\\script.py'
```

E usar a função: **`os.path.basename(path)`**

Ele retorna: 'script.py', que é o nome de base.

Já se usar: **`os.path.dirname(path)`**

Ele retorna: "C:\\Estudo\\Python", que é o nome do diretório do arquivo da base.

Se quiser os dois, use a função: **`os.path.split(path)`**

Que ela se torna uma [tupla](#) com os dois caminhos: ('C:\\Estudo\\Python', 'script.py'), prontinho pra você usar as duas informações.

- **Como Calcular o tamanho de um arquivo: `getsize()`**

Tem uma função bem bacana no módulo **`os`**, chamada **`getsize`**, que como o nome diz, pega o tamanho do arquivo.

Se o nome do script que você está programando é **arquivos.py**, o código abaixo vai printar o tamanho em bytes desse arquivo, em seu sistema operacional:

```
import os
print( os.path.getsize('arquivos.py') )
```

Aqui deu 50, e aí?

- **Exibindo conteúdo de uma pasta ou diretório: listdir()**

Se ao invés do endereço de um arquivo, você passar o caminho de uma pasta ou diretório, dessa vez pra função **listdir**, ele printar na sua tela todo o conteúdo daquela pasta.

Se usar Windows, teste o seguinte código:

```
import os
print( os.listdir('C:\\Windows\\System32') )
```

Se usar Linux, teste o seguinte código:

```
import os
print( os.listdir('/') )
```

Interessante e simples, não é?

São os arquivos mais importantes de seu sistema operacional, e agora estão sob seu poder...Mwahahahaha.

Não se assuste, programador é assim, tem poderes que os outros não tem. Tem acesso, tem conhecimento e possibilidades que mais ninguém tem.

- **Exercício de Arquivos em Python**

Crie um script em Python que liste todos os arquivos e diretórios de um determinado caminho, e calcule a soma de todos os arquivos em bytes.

Vamos armazenar os bytes na variável **tamanho**.

A função **listdir** uma lista de nomes de arquivos.

Então, podemos usar um laço for, com uma variável temporária **filename**, que vai assumir o nome de cada arquivo desses, dessa lista. Usamos a função **os.path.join()** para obter o endereço completo de cada arquivo e usamos a **getsize** para obter o tamanho de cada um e somar na variável **tamanho**.

Veja como ficou nosso código Python

```
import os

tamanho = 0
for filename in os.listdir('C:\\Windows\\System32'):
    tamanho += os.path.getsize(os.path.join('C:\\Windows\\System32',
filename))

print(tamanho)
```

- **Módulo OS**

Existe ainda, uma série de outras funções e possibilidades, para se trabalhar com pastas e arquivos, pelo fantástico módulo **os**.

Recomendamos que, sempre que tiver alguma dúvida, consulte a documentação oficial da linguagem de programação Python:

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

Como Ler e Escrever num mesmo arquivo ao mesmo tempo

- **Exercício: Ler e Escrever num arquivo**

"Crie um programa simples que pergunta se a pessoa deseja ler um arquivo ou escrever algo nele."

Nosso script vai funcionar assim:

Aparece um menu de opções (sair, ler ou escrever)

Se digitar ler, lê o conteúdo do arquivo

Se optar por escrever, escreve algo no arquivo

Isso em um looping indefinido, acaba quando o usuário quiser.

Primeiro de tudo, importamos o módulo **os.path**, isso vai ser necessário pois vamos usar a função **isfile()** para checar se o arquivo teste.exe existe

Se retornar false, é porque não existe, então cai dentro de um IF que vai criar esse arquivo (usando o modo de abertura de escrita **w**).

Agora que criamos, vamos abrir o arquivo no modo de leitura e escrita: **r+**

Arquivo aberto, perguntamos que opção ele deseja fazer.

Se for 0, **op** fica nulo e termina o laço WHILE.

Se for 1, o usuário quer ler o que tem no arquivo.

Para isto, basta usar a função **read()** dentro de um print.

Em seguida, fechamos o arquivo, já que terminamos de usar.

Se ele digitar 2, vai cair na opção de escrever.

Então pedimos um número, *truncamos* o arquivo (retiramos o que tem nele), escrevemos com o método **write()** e fechamos o arquivo!

E prontinho, é só isso, veja como ficou nosso código Python:

```
import os.path
```

```
op=1
```

```

while op:
    if os.path.isfile("teste.txt") is False:
        print("Arquivo teste.txt nao existe. Criando...")
        meuArquivo = open("teste.txt", "w")

    meuArquivo = open("teste.txt", "r+")

    op=int(input("0. Sair \n"
                "1. Ler\n"
                "2. Escrever\n"))

    if op==1:
        print( meuArquivo.read() )
        meuArquivo.close()

    if op==2:
        num = input("Numero:")
        meuArquivo.truncate()
        meuArquivo.write(num)
        meuArquivo.close()

meuArquivo.close()

```

• Exercício de Arquivos em Python

"Ao escrever compare o número que o usuário digitou com o número que está lá armazenado. Só escreva no arquivo se o usuário digitar um número MAIOR do que aquele que está lá".

A única novidade, vai ser a variável **num2**, ela vai ler o que tem no arquivo, pelo método **read()**.

Em seguida, vamos escrever no arquivo apenas em duas condições:

Se o arquivo estiver vazio, ou seja, num2="

Se o número do usuário for maior que o armazenado: `int(num) > int(num2)`

Lembre-se de passar **num** e **num2** para inteiro, pois em arquivos, elas são strings!

Se nenhuma dessas condições for atendida, nada escrevemos no arquivo e dizemos isso dentro de um print no ELSE.

Veja como ficou nosso código:

```
import os.path

op=1
while op:
    if os.path.isfile("teste.txt") is False:
        print("Arquivo teste.txt nao existe. Criando...")
        meuArquivo = open("teste.txt", "w")

    else:
        meuArquivo = open("teste.txt", "r+")

    op=int(input("0. Sair \n"
                "1. Ler\n"
                "2. Escrever\n"))

    if op==1:
        print( meuArquivo.read() )
        meuArquivo.close()

    if op==2:
        num = input("Numero:")
        num2 = meuArquivo.read()
        meuArquivo.seek(0)

        if (num2 is " ") or (int(num)>int(num2)):
            meuArquivo.truncate()
            meuArquivo.write(num)
            meuArquivo.close()
        else:
            print(num," menor que ", num2)
    meuArquivo.close()
```

Exercício Proposto

No artigo [Como criar um jogo em Python](#), fizemos um game onde o computador sorteia um número de 1 até 100 e você deve adivinhar.

Implemente a função **record** nele.

Ou seja, num arquivo de no record.txt você deve armazenar o recorde do número mínimo de tentativas que alguém conseguiu acertar!

Resolução:

Quando o usuário acerta, nosso código chama a função **record()** que recebe o número de tentativas que o usuário tentou.

Primeiro, testamos se o arquivo record.txt existe.

Se não existe, criamos.

Depois, abrimos ele com modo de operação de leitura e escrita "r+".

Lemos o que tem lá dentro e armazenamos em **num2**.

Usamos a função **seek(0)** para posicionar o ponteiro no início (posição 0), pois foi lá pro final ao lermos com a **read()**.

Se o arquivo for vazio ou o número de tentativas for menor do que o record armazenado em *record.txt*, damos os parabéns dizendo que ele bateu o record, escrevemos o record no arquivo (passamos antes pra string, usando função **str()**, pois o **write()** só recebe string) e fechamos o arquivo.

Se nenhuma dessas condições forem aceitas, é porque você não bateu o record e cai no ELSE, dando essa mensagem.

Código Python:

```
import random
```

```
import os.path
```

```
def gera():
```

```
    return random.randint(1,100)
```

```
def record(tentativa):
```

```
    if os.path.isfile("record.txt") is False:
```

```
        print("Arquivo record.txt nao existe. Criando...")
```

```
        meuArquivo = open("record.txt", "w")
```

```
    meuArquivo = open("record.txt", "r+")
```

```
    num2 = meuArquivo.read()
```

```
    meuArquivo.seek(0)
```

```
    if (num2 is "") or (tentativa<int(num2)):
```

```
        print("Parabéns! Você bateu o recorde")
```

```
        meuArquivo.truncate()
```

```
        meuArquivo.write(str(tentativa))
```

```
        meuArquivo.close()
```

```
    else:
```



```
print("O recorde ainda é de ", num2, " tentativas")
```

```
def game():
```

```
    resposta = gera()
```

```
    tentativa = 0
```

```
    print("\nPalpite gerado!")
```

```
    chute=0
```

```
    while chute is not resposta:
```

```
        tentativa +=1
```

```
        chute = int(input("Qual seu chute: "))
```

```
        if chute > resposta:
```

```
            print("Errou! É um valor menor que ", chute)
```

```
        elif chute < resposta:
```

```
            print("Errou! É um valor maior que ", chute)
```

```
        else:
```

```
            print("Parabéns! O número gerado foi ",resposta, \
                  "\nAcertou em ",tentativa," tentativas!")
```

```
            record(tentativa)
```

```
while True:
```

```
    game()
```

Strings

Junto dos números, os textos são a forma de comunicação mais importantes dos seres humanos.

Você está lendo nosso **Curso de Python** através desse texto.

Você lê um livro, um site, uma carta, um boleto, o letreiro de um ônibus, a legenda de um filme...tudo via texto, ou seja, *strings*.

O sistema operacional Linux, é todo código-fonte aberto, ou seja, um amontoado de *strings*.

As configurações de iluminação do seu notebook, tamanho da tela, ícones...tudo informação em texto.

Entrou no seu jogo favorito, e tá lá a fase que você estava, os items e pontos que tinha antes de desligar pela última vez? Informações salvas, tudo na forma de texto.

Sabe o corretor automático do WhatsApp? Do Word?

O localizar um texto? Substituir por outro? Formatar, ajusta, reduzir tamanho de letra, fonte...?

Tudo texto. Tudo trabalho com *string*. E é isso, e muito mais, que vamos aprender nesta seção de **String** de nosso curso de Python.

String - O que é ? Como funciona ? Onde se usa ?

- **Strings** - O que são ?

Esta seção de nosso **Curso de Python**, poderia muito bem estar na de [Sequências](#), pois é simplesmente isso que uma string é: uma sequência de caracteres.

E por caracteres, entendam:

Letras maiúsculas

Letras minúsculas

Acentos: ' ` ~ ^

Caracteres especiais: ! @ # \$ % " & * ()

E outros próprios da computação: como quebra de linha '\n' e um tab '\t'

Ou seja, qualquer representaçãozinha de algo que você possa, de algum modo ver ou perceber, é um caractere. O caractere é a maneira do computador, da máquina, se comunicar com a gente.

Lembre-se: eles entendem apenas bits (correntes e voltagens elétricas), binário, 1 e 0...a gente 'se dá melhor' com *strings*.

- **Strings** - Como funciona? O que fazer com elas?

Todo e qualquer trabalho que você possa imaginar, vai usar *strings*.

Imagine o Google...as pessoas digitam strings, eles devem armazenar trilhões de strings sobre sites, informações, tem que achar dados, bater com o que a pessoa busca, baseado nos gostos de cada um, sugerir outros sites, vídeos...tudo isso é tratamento e trabalho intenso com *strings*.

Um sistema federal, com nomes, CPF, RG, CNH, lista com ficha criminal, ficha médica...tudo são dados na forma de *strings*. Achar alguém, bater uma informação com outra, é tudo comparar string.

Vamos estudar coisas como:

- Saber se um caractere é maiúsculo ou minúsculo
- Saber se foi digitado um texto ou um número (ex: não pode digitar número no campo de CEP)

- Passar tudo pra minúsculo (ex: endereço de sites e e-mail, tem que passar tudo pra minúsculo)
- Localizar e substituir dos editores de texto é fazer procura em strings, apagar trechos, substituir por outros etc
- Aplicar um corretor ortográfico (comparar suas strings com a de um dicionário, de sua língua)
- Traduzir um texto entre línguas
- Acessar o *clipboard* usando o Python, região que fica armazenado o copiar e colar do seu computador

E por ai vai, os usos e possibilidades com *strings* são simplesmente ilimitados.

No próximo tutorial vamos rever como usar, como criar, acessar seus elementos, exibir alguns caracteres especiais bem como descobrir o tamanho de uma string com a função **len**.

Como usar, Acessar Caracteres e Saber Tamanho de String

- **Como Usar Strings**

Vamos dar uma relembração nos conceitos principais de string.

Representamos uma string por:

- 'algo'
- "algo"

Ou seja, qualquer caractere entre aspas simples ou aspas duplas.

É errado fazer: 'algo" ou "algo'

Se começou com simples, termina com simples. Se começa com dupla, termina em aspas duplas.

É possível também usar as dois, para mostrar aspas simples ou dupla num print, por exemplo:

'Curso "Python" Progressivo'

A saída é: Curso "Python" Progressivo

Isso ocorreu porque o Python entende que tudo dentro das aspas simples, era uma string, incluindo as aspas duplas.

- **Caracteres Especiais em Python**

Para representar coisas diferentes de texto, número e acentuação, usamos a barra de escape, que nos permite utilizar os seguintes caracteres especiais:

\\ - Exibe uma barra

\' - Exibe a aspa simples

\" - Exibe a aspa dupla

\a - Dá um bipe

\b - Retrocesso

\f - Avanço

\n - Caractere de nova linha (enter)

\r - Carriage return

\t - Tab horizontal

\n - Tab vertical

Acessando caracteres de uma string

Podemos acessar caractere de uma string usando o laço for:

```
texto = "Curso Python Progressivo"
```

```
for caractere in texto:  
    print(caractere)
```

Ou seja, a variável **caractere** vai assumindo o valor de cada caractere da string *texto*, um por um e imprimindo.

Para exibir sem a quebra de linha:

```
texto = "Curso Python Progressivo"
```

```
for caractere in texto:  
    print(caractere, end="")
```

Você também pode acessar diretamente caractere por caractere, como se fosse uma lista:

```
texto[0]='C'  
texto[1]='u'  
texto[2]='r'  
texto[3]='s'  
texto[4]='o'  
texto[5]=' '  
etc
```

Já o comando:

texto[0:5] imprime 5 caracteres, de posição 0, 1, 2, 3 e 4: 'Curso'

O comando:

texto[6:] - exibe do índice 6 em diante: 'Python Progressivo'

- **Tamanho de uma string: len()**

Uma função nativa do Python e bem útil, é a **len()**.

Você chama ela com uma string dentro, e ela te retorna o número de caracteres.

Fazendo:

`print(len(texto))` - retorna o valor 24, pois tem vinte e quatro caracteres.

- **Concatenar Strings: operador +**

Vejam o seguinte código:

```
texto = "Curso"
```

```
print(texto)
```

```
texto = texto + " Python Progressivo"
```

```
print(texto)
```

Inserimos a string 'Curso' na variável *texto*.

Em seguida, fazemos essa variável se igualar com ela mesma MAIS a string " Python Progressivo".

Isso foi feito com sinal de +, é o operador de concatenação, ou seja, vai unir uma string na outra.

O leitor mais atento, vai pensar:

"Mas você disse que strings são imutáveis"

De fato, são.

A variável *texto* apontava para um endereço de memória que tinha "Curso" lá.

Depois, passou a apontar para outro endereço, que tem a string "Curso Python Progressivo".

A antiga referência vai deixar de ser usada e portanto descartada de imediato pelo Python.

Strings Maiúsculas e Minúsculas em Python

- Transformar string em maiúscula ou minúscula: **upper()** e **lower()**

O método **upper()** recebe o valor de uma string e retorna ela mesma, mas com todos os caracteres em maiúscula.

Esse método faz parte dos objetos do tipo string.

Então para transformar uma string *texto* em maiúscula, fazemos:

texto.upper()

Veja um exemplo:

```
texto = "Curso Python Progressivo"  
print(texto)
```

```
texto = texto.upper()  
print(texto)
```

Resultado:

```
Curso Python Progressivo  
CURSO PYTHON PROGRESSIVO
```

O mesmo vale para transformar todos os caracteres da string em minúsculos, usando o método **lower()**:

```
texto = "Curso Python Progressivo"  
print(texto)
```

```
texto = texto.lower()  
print(texto)
```

Teste e veja o resultado.

- **String toda em maiúscula ou minúscula: `isupper()` e `islower()`**

Para detectar se uma string é toda maiúscula ou toda minúscula, usamos o método **`isupper()`** que retorna True se todos caracteres forem maiúsculos, e False se não forem.

Vamos testar se a string 'CURSO PYTHON PROGRESSIVO' é toda maiúscula:

Experimente colocar apenas uma letrinha em minúscula, e vai cair no ELSE.

Para saber se toda a string é minúscula, use o método **`islower()`**, que é semelhante.

- **Exercício de String em Python**

Crie um script que peça uma string ao usuário e diga se:

Ela é toda maiúscula

Ela é toda minúscula

Tem caracteres maiúsculos e minúsculos

No primeiro teste condicional, verificamos se tudo é maiúsculo.

Se for, ok, avisa isso.

Se não for, cai no ELIF que vai testar se é tudo minúsculo.

Se for, avisa que é.

E por fim, se não for tudo maiúsculo ou tudo minúsculo, é porque ta misturado:

```
while True:
    texto = input("Digite uma string: ")

    if texto.isupper():
        print("Tudo maiusculo")
    elif texto.islower():
        print("Tudo minusculo")
    else:
        print("Misturado")
```

Strings com Letras, Números e/ou Caracteres Especiais

- É tudo letra? **isalpha()**

O método **isalpha()** retorna True se todas as letras de uma string forem letras.

Ou seja, se tiver algum número ou caractere especial.

Ela também retorna False se estiver vazia.

O código a seguir pede uma string ao usuário e diz se é tudo letra ou misturado:

```
while True:
    texto = input("Digite uma string: ")

    if texto.isalpha():
        print("Tudo letra")
    else:
        print("Nem tudo é letra")
```

- É tudo número ? **isdecimal()**

Já a **isdecimal()** retorna True se tudo for número.

Se tiver alguma letra, caractere especial ou a string for vazia, retorna False. O código abaixo identifica se é tudo letra, tudo número ou misturado:

```
while True:
    texto = input("Digite uma string: ")

    if texto.isalpha():
        print("Tudo tudo letra")
    elif texto.isdecimal():
        print("Tudo numero")
    else:
        print("Misturado, vazio ou caractere especial")
```

- **Letras e Números ? `isalnum()`**

Essa retorna True se for composta somente por letras e números.

Retorna False se for vazia ou tiver algum caractere especial.

O código abaixo diz se sua string é tudo número, tudo letra, letra e número ou tem caractere especial ou vazia:

```
while True:
    texto = input("Digite uma string: ")

    if texto.isalpha():
        print("Tudo tudo letra")
    elif texto.isdecimal():
        print("Tudo numero")
    elif texto.isalnum():
        print("Numeros e letras")
    else:
        print("Vazia ou caractere especial")
```

- **Só espaço, tabulação ou quebra de linha: `isspace()`**

Já o método `isspace()` só retorna True se tivermos espaço em branco, tabulação ou quebra de linha.

Retorna False caso contrário ou a string seja vazia.

```
while True:
    texto = input("Digite uma string: ")

    if texto.isalpha():
        print("Tudo tudo letra")
    elif texto.isdecimal():
        print("Tudo numero")
    elif texto.isalnum():
        print("Numeros e letras")
    elif texto.isspace():
        print("Composto so de espaço, e/ou tabulação e/ou quebra de
linha")
    else:
        print("Vazia ou caractere especial")
```

Como Unir e Separar Strings: `join()` e `split()`

- **Unir Strings: Método `join()`**

Vamos supor que temos a lista:

```
lista = ['Curso','Python','Progressivo']
```

Se usarmos: `','.join(lista)`

Ele vai unir cada string da lista, colocando uma vírgula entre as strings.

O resultado é uma string só:

```
"Curso,Python,Progressivo"
```

É mais interessante unir essas palavras com espaço em branco.

Então fazemos: `' '.join(lista)`

O resultado é a string: "Curso Python Progressivo".

Teste o código:

```
lista=['Curso','Python','Progressivo']
```

```
#Unindo as palavras com vírgula
```

```
print( ','.join(lista) )
```

```
#Unindo as palavras com espaço
```

```
print( ' '.join(lista) )
```

Ou seja, o comando: `ch.join(lista)`

Ele vai pegar cada elemento da lista *lista* e grudar um no outro com o caractere *ch* unindo eles.

- **Como Separar e Quebrar Strings: `split()`**

Assim como temos método para unir, temos para separar.

Por exemplo, para transformar a string: "Curso Python Progressivo)" em palavras separadas, quebrando onde tem um espaço em branco, fazemos:

`lista.split()`

O resultado é a lista: `['Curso','Python','Progressivo']`

Teste:

```
texto="Curso Python Progressivo"  
print( texto.split() )
```

Note que ele quebrou automaticamente nos espaços em branco.
Mas podemos quebrar em qualquer caractere.

Por exemplo, vamos supor que o Silvio Santos nos deu a seguinte string:
"123PI567PI9..."

Fazendo: **texto.split('PI')**

Ele vai arrancar os 'PI' da string e devolve uma lista com elementos separados onde antes era 'PI'.

Teste:

```
texto="123PI567PI9..."  
print( texto.split('PI') )
```

O resultado é a lista:

```
['123', '567', '9...']
```

Buscar e Substituir substring em String

- **Localizar no final da string: `endswith(substring)`**

Usamos o método **`endswith()`** (do inglês 'termina com') para localizar uma substring (passada como argumento para o método) dentro de uma string maior, da seguinte maneira:

- `texto.endswith(substring)`

Se a string *substring* estiver ao final da string *texto*, esse método retorna `True`.

Por exemplo, vamos criar um script que pede ao usuário o nome e extensão de um arquivo.

Se terminar com `.txt`, `.pdf` ou `.png`, dizemos se é um arquivo de texto, um PDF ou uma imagem.

```
while True:
```

```
    texto = input("Digite o nome de um arquivo com sua extensão:" )
```

```
    if texto.endswith('.txt'):
```

```
        print("É um arquivo de texto")
```

```
    elif texto.endswith('.pdf'):
```

```
        print("É um arquivo do Acrobat Reader")
```

```
    elif texto.endswith('.png'):
```

```
        print("É uma imagem")
```

```
    else:
```

```
        print("Outro tipo de arquivo")
```

- **Buscar no início da string: `startswith(substring)`**

Analogamente, é possível verificar se alguma coisa está (uma string) está no começo, bem no início de outra string, através do método **`startswith`** (do inglês 'começa com'):

- `texto.startswith(substring)`

Esse método retorna **`True`** se *substring* está no começo da string *texto*.

Por exemplo, vamos supor que você ficou encarregado de convocar todos os nomes 'Francisco' da escola ou empresa que trabalha. É muito simples,

basta pegar todas as strings (de uma lista ou via **readlines()** de um arquivo) e usar:

- `texto.startswith('Francisco')`

Prontinho, quem for Xico, vai ser identificado e devidamente selecionado com este simples e prático método.

- **Achar qualquer substring em qualquer lugar de uma string:**
find()

Se quisermos achar algo no começo, usamos **startswith**.

Se quisermos buscar algo no final, usamos **endswith**.

Se quisermos algo em qualquer local da string? Até mesmo no começo ou final, ou seja, não sabemos onde a substring pode estar?

No exemplo anterior: `texto.startswith('Francisco')`

Só selecionaria aqueles cujos nomes começam por Xico.

Mas e se o nome da criatura for José Francisco Silva ?

Não encontra!

Mas se usar: `texto.find('Francisco')`

Ela vai retornar o menor índice da primeira aparição dessa substring.

Se não encontrar, retorna -1.

- **Substituir substring por outra: **replace()****

Ok, aprendemos a encontrar qualquer string, em qualquer local de outra string.

Mas se ao invés de localizar, queiramos substituir?

Certamente você já usou o 'localizar e substituir' do Word.

Em Python, usamos o método **replace**, que recebe dois argumentos. Seja *texto* a string:

`texto.replace(nova,antiga)`

Ou seja, toda vez que achar a substring *antiga* ele vai colocar a substring *nova*, na string *texto*.

Esse método retorna uma nova string, com o texto substituído.

Exercício de substituição de string

"Você criou um software na sua escola, onde você pergunta a opinião dos alunos sobre a linguagem de programação Python e tem que apresentar esse texto pro dono da escola.

Porém, os alunos escrevem muito 'Pyton' ao invés de 'Python'. E também usam muito 'eh' ao invés de 'é'.

Crie um script pra você, que faça essas substituições."

O código, bem simples de ser entendido, é:

```
texto = input("O que acha do curso: " )
texto=texto.replace('Pyton','Python')
texto=texto.replace('eh','é')
print("Texto corrigido  :",texto)
```

Veja o resultado, que bacana:

```
O que acha do curso: Pyton eh legal
Texto corrigido   : Python é legal
```

Agora você já pode criar um corretor automático de textos pra você :)

Regex, parte 1

- **Expressão Regular: O que é? Para que serve ?**

Vamos supor que você baixou um site inteiro da internet, e vai tratar de milhões e milhões de linha de texto nele, em busca de algo específico: números de telefone.

Não tem como da rum control+F ou usar métodos para [buscar substrings](#), por um detalhe importante: você **não sabe** que números são esses.

Mas você sabe de uma coisa: começam por parêntesis, tem dois dígitos (o DDD), fecha parêntesis, tem um 9, depois mais quatro dígitos, as vezes um hífen as vezes não, e mais quatro números.

E é aí que as expressões regulares vão atuar: você vai definir um **padrão** e elas vão buscar no texto esse padrão. No parágrafo anterior acabamos de criar um padrão para achar números de telefone.

É uma expressão regular, no caso, ainda em nossa cabeça, já já vamos transformar ela em código Python.

Agora e-mail, vamos supor que queira 'caçar' e-mail na grande rede para fazer propaganda de sua empresa. Os endereços de e-mail tem o seguinte formato:

- 1.algum texto
- 2.@
- 3.algum site
- 4..com em algum lugar (como gmail.com ou bol.com.br, mas tem '.com')

Pronto, criamos um padrão. Com isso, podemos criar uma expressão regular para tratar um texto gigante, que vai nos retornar os endereços de e-mail!

As *regular expressions* (também conhecidas pela abreviação *Regex*) são isso: padrão de um texto que vamos procurar.

- **Trabalhando sem expressão regular**

Primeiro, vamos resolver um problema sem expressão regular, para você entender o que é criar um padrão para detectar algo.

"Crie uma função que recebe um argumento do tipo string, e diz se ela representa um telefone do tipo xxxx-yyy ou não".

De cara, nossa *string* tem que ter tamanho 9. Usamos a função **len()** pra isso. Se não tiver, já retorna falso, não tá do jeito que a gente quer.

Depois, os caracteres das 4 primeiras posições.

Eles devem ser números, ou seja, vamos usar a função **isdecimal()** para definir se são ou não.

Já o quinto caractere, de índice 4, deve ser o hífen '-'.
Se não for, mostramos mensagem de erro.

Por fim, testamos novamente se os 4 últimos são números.

Nosso código fica assim:

```
def eTelefone(numero):  
    if len(numero) != 9:  
        print("Tamanho diferente de 9")  
        return False  
  
    for index in range(0,4):  
        if numero[index].isdecimal() is not True:  
            print("4 primeiros nao sao numero")  
            return False  
    if numero[4] != '-':  
        print("Nao tem hífen")  
        return False  
  
    for index in range(5,9):  
        if numero[index].isdecimal() is not True:  
            print("4 ultimos nao sao numero")  
            return False  
  
    return True  
  
while True:  
    numero = input("Numero no formato 'xxxx-yyy': ")  
  
    if eTelefone(numero) is True:  
        print("Válido")
```

```
else:  
    print("Inválido")
```

Experimente!

Teste:

1234-6789

12345678

1234 5678

Bacana, não ?

Mas isso só funciona se você digitar apenas um número, e se digitar:
"Fala galera, meu numero é 1234-5678"

Não detecta!

Aí é simples, temos que analisar blocos de 9 caracteres, 9 em 9.
Fazemos isso assim: pedaco[i:i+9]

Onde i varia de 0 até o tamanho da string, só pra garantir que vamos percorrer todos os blocos possíveis de 9 caracteres.

Veja como fica nosso código que detecta qualquer número nessas condições, numa string qualquer:

```
def eTelefone(numero):  
    if len(numero) != 9:  
        return False  
    for index in range(0,4):  
        if numero[index].isdecimal() is not True:  
            return False  
    if numero[4] != '-':  
        return False  
    for index in range(5,9):  
        if numero[index].isdecimal() is not True:  
            return False  
    print("Numero detectado: ", numero)  
    return True  
  
while True:  
    numero = input("Numero no formato 'xxxx-yyy': " )
```

```
for i in range( len(numero)):
    pedaco=numero[i:i+9]
    eTelefone(pedaco)
```

Veja o resultado de alguns testes:

```
Numero no formato 'xxxx-yyy': 1234-5678
Numero detectado: 1234-5678
Numero no formato 'xxxx-yyy': oi meu numero eh 1111-5555
Numero detectado: 1111-5555
```

- **Usando Expressão Regular**

O código anterior é bacaninha, mas tem vários problemas:

É longo

Se alguém digitar xxxx-yyy, xxxxyyyy, ou (xx) yyyzzzz ou xx yyyy-zzzz etc etc, dará erro, e na verdade são números válidos de telefone

Se eu repetir: xxxx-yyy xxxx-yyy varias vezes, vai detectar e repetir várias vezes

E por aí vai, de problemas.

É aí que entra em ação nossas *regexes*.

Regex de dígito: `\d`

A expressão regular mais simples é o padrão `\d`

Ele representa um simples dígito. Apenas isso.

Qualquer numeral de 0 até 9, é detectado via `"\d"`

Logo uma regex que detecta nosso número de telefone é:

`\d\d\d\d-\d\d\d\d`

Se colocarmos um valor entre chaves, dizemos que aquela regex deve 'bater' aquele número de vezes.

Note que repetimos `\d` quatro vezes, um hífen e mais quatro vezes repetimos `\d`.

Podemos simplificar para: `\d{4}-\d{4}`

Bem mais simples, não ?

E com DDD ? As expressões abaixo funcionariam

`\d{2} \d{4}-\d{4}` :bate com xx yyyy-zzzz

`(\d{2}) \d{4}-\d{4}`: bate com (xx) yyyy-zzzz

`(\d{2}) \d{8}` : bate com xx yyyyzzzz

Tá sentindo a alma do negócio?

- **O Módulo `re`**

O Python tem uma coleção de funções para tratarmos e lidarmos com *regex*, estão todas no módulo **`re`**.

Ou seja, faça sempre a importação:

- `import re`

- **Objetos *Regex* e função `recompile`**

A função **`recompile()`** do módulo **`re`** recebe uma string com sua regex e retorna um objeto do tipo *Regex*.

Vamos passar uma expressão regular para a função *recompile* e armazenar o retorno na variável *minhaRegex*:

- `minhaRegex = re.compile(r'\d{4}-\d{4}')`

Pessoal, **`\d`** representa qualquer caractere que é um dígito.

Diferente do **`\n`** que representa uma única coisa, **`\d`** são duas coisas: uma barra e um d

Usamos a letra **`r`** antes da string no método *compile* para não precisar fazer:

- `minhaRegex = re.compile("\\d{4}-\\d{4}')`

Lembre-se: para representar uma barra **`\`** em uma string, temos que escapar ela com outra barra: **`\\`**

Usando o **`r`**, podemos escrever diretamente sem precisa escapar.

- **Métodos *search* e *group* dos Objetos *Regex***

Agora temos uma variável do tipo *Regex*, em *minhaRegex*.

Esses objetos tem um método chamado **search()**, que vai receber uma string e pesquisar nela toda e qualquer correspondência da regex que você criou. Se não achar, ela retorna *None*.

Vamos armazenar a string que o usuário vai digitar em **texto** e guardar as correspondências que bateram na variável **resultado**:

- `resultado = minhaRegex.search(texto)`

O método **search** retorna um objeto do tipo **Match** (algo como 'bater', no sentido de combinou, deu certo...*deu match* no Instagram). Ou seja, a variável **resultado** é um objeto do tipo *Match*.

Objetos do tipo *Match* tem um método chamado **group()**, que armazena todas as correspondências da regex encontradas na string.

O código, com expressão regular, que acha um número agora é:

```
import re
while True:
    texto = input("Numero no formato 'xxxx-yy': ")

    minhaRegex = re.compile(r'\d{4}-\d{4}')
    resultado = minhaRegex.search(texto)

    print(resultado.group())
```

- **Achar todas ocorrências: findall**

No código anterior, tem um problema: só vai achar a primeira ocorrência que bater com a regex.

Para achar todas, usamos o método **findall**.

Ele recebe duas strings: o padrão da regex, e a string.

E simplesmente retorna uma lista com todas as ocorrências que batem.

Veja com fica o código que acha tudo:

```
import re

while True:
    texto = input("Numero no formato 'xxxx-yy': ")
    minhaRegex = re.findall(r'\d{4}-\d{4}', texto)
    print(minhaRegex)
```

Grupos e Parêntesis ()

- **Grupos em Regex: ()**

No tutorial anterior, de [Regex parte 1](#), já vimos o que é uma regex, para que serve, onde são usadas e trabalhamos com um exemplo: os dos telefones do tipo xxxx-yyy, onde usamos a regex `\d` para representar qualquer dígito.

Agora vamos um pouquinho mais além, e trabalhar com números que usam também o DDD.

Esses números de telefone, tem o seguinte padrão:

xx yyy-zzz

Uma regex que capta esse padrão, seria:

- `'\d\d \d\d\d\d-\d\d\d\d'`

Mais concisamente:

- `'\d{2} \d{4}-\d{4}'`

Note que tem 4 coisas aí: o DDD, o primeiro bloco de dígitos (que geralmente você descobre a operadora por ele) e o último bloco.

Vamos organizar esses blocos em **grupos**. Para isso, colocamos eles entre parêntesis:

- `'(\d{2}) (\d{4})-(\d{4})'`

- **Como Usar os Grupos em Regex: `group()`**

Tenha uma coisa em mente: os parêntesis não vão alterar sua regex. Ou seja, não precisa ter parêntesis ao redor dos números em sua string, pra dar *match*.

Então para que serve? É simples: podemos acessar diretamente cada grupo depois, pelo método **group**.

Ao dar *match*, o primeiro grupo poderá ser acessado pelo método: `group(1)`

O segundo grupo é acessado por: `group(2)`

E o terceiro por: `group(3)`

O código abaixo detecta um número de telefone no formato: xx yyyy-zzzz e mostra, separadamente, o DDD desse número:

```
import re
while True:
    texto = input("Digite sua string: ")

    minhaRegex = re.compile(r'(\d{2}) (\d{4})-(\d{4})')
    resultado = minhaRegex.search(texto)

    print("Numero detectado:", resultado.group())
    print("DDD desse numero:", resultado.group(1))
```

O método **group** sem argumento algum, já havíamos utilizado: ele mostra tudo, toda a *regex* que bateu. Usamos ele pra exibir o número inteiro do telefone, em seguida usamos o primeiro grupo, via **group(1)**, para exibir somente o DDD.

Prontinho, além de termos extraído o número de telefone, fomos mais além e ainda detectamos que trecho é o de DDD, agora você pode extrair milhões de números de um arquivo de texto, site, documento etc, e ainda categorizar, dizer quantos são de cada cidade, tratando o DDD.

Se desejar uma [tupla](#), ou seja, uma lista com todos os grupos que foram detectados, use o método **groups()** (sem argumento nenhum), que você vai ter a tupla separada bonitinha, separando os grupos de maneira organizada.

O caractere pipe | (OU em Regex)

- **Caractere Pipe: |**

Existe um caractere especial no mundo das regex, é a barra vertical: |
Ela é chamada de *pipe* e tem um significado de OU: *ou* isso *ou* aquilo.

Usamos elas em expressões regulares, quando queremos encontrar um padrão **OU** outro, ou seja, qualquer um ou mesmo os dois.

Exemplo de uso:

- `r'padrao1 | padrao2 | padrao3'`

Essa expressão regular vai em busca de qualquer um dos três padrões.
Por exemplo, script abaixo detecta se o alguma das bandas favoritas do usuário é Iron Maiden, Rush ou Deep Purple:

```
import re
```

```
while True:
```

```
    texto = input("Quais suas bandas favoritas: " )
```

```
    minhaRegex = re.findall(r'Rush|Iron Maiden|Deep Purple', texto)
```

```
    print("Gostei dessas: ", minhaRegex)
```

Não digite nenhuma banda de forró, funk ou k-pop, o programa vai travar e vai queimar seu HD. E sua casa vai pegar fogo também.

- **Exercício de Regex Usando Pipe |**

No artigo passado, no [tutorial sobre grupos e parêntesis em regex](#), estávamos em nossa saga em criar um script que vai detectar um número de telefone em uma string qualquer (pode escrever até a bíblia nessa string...provavelmente você não vai achar nenhum número de telefone na bíblia...um trecho de uma música do Iron Maiden, talvez...).

Porém, um número pode ter os seguintes formatos:

- `xx yyyy-zzzz`
- `xx yyyyzzzz`
- `xyyyyyzzzz`

Veja bem...pode ser um OU outro OU outro...OU, entendeu?

Sim, vamos usar o caractere de *pipe* para detectar qualquer um desses tipos de números.

xx yyyy-zzzz, representamos por: `\d{2} \d{4}-\d{4}`

xx yyyy-zzzz, representamos por: `\d{3} \d{4}\d{4}` ou simplesmente: `\d{3} \d{8}`

xyyyyyzzzz, representamos simplesmente por: `\d{10}`

E prontinho. Dentro da regex, separamos cada uma desses padrões pelo caractere de pipe |, e as regex em Python fazem o resto, vão buscar qualquer um desses três padrões.

Veja como fica nosso código:

```
import re

while True:
    texto = input("Digite sua string: " )

    minhaRegex = re.findall(r'\d{2} \d{4}-\d{4}|\d{2} \d{8}|\d{10}', texto)
    print("Gostei dessas: ", minhaRegex)
```

Olhe como nossos scripts detectores de números de telefone estão ficando cada vez mais eficientes!

E o mais bacana: estamos fazendo cada vez mais, escrevendo cada vez menos código.

Essa é a beleza da programação em Python.

Exercício de Regex

Faça um script que detecte números no formato:

(xx) yyyy-zzzz

(xx) yyyzzzzz

Para detectar o parêntese, você tem que *escapar* ele: `\(` e `\)`, ok ?

```
import re

while True:
    texto = input("Digite sua string: " )
    minhaRegex = re.findall(r'\(\d{2}\) \d{4}-\d{4}|\(\d{2}\) \d{8}', texto)
    print("Gostei dessas: ", minhaRegex)
```

Ponto de Interrogação, Asterisco e Sinal de Adição em Regex

- **Interrogação em Regex: ?**

Usamos o símbolo de interrogação ? quando queremos que um determinado padrão seja opcional.

Ou seja, se tiver lá, ok. Se não tiver, ok também.

A regex é assim:

- `r'padrao1 (padrao2)? padrao3'`

Ou seja, vai em busca do *padrao1*, tem que ter ele.

Já o *padrao2* é um grupo opcional, não é obrigatório que esteja lá, pois usamos o caractere ?

Por fim, busca o *padrao3*.

Por exemplo, vamos fazer script que detecta os seguintes formatos de número de telefone:

`xx yyyy-zzzz`

`xx yyyyzzzz`

Ou seja, o hífen é opcional. Tanto faz se o usuário digitar ou não.

Se é tanto faz, temos que usar o caractere de interrogação ?

Veja como fica o código:

```
import re
```

```
while True:
```

```
    texto = input("Digite sua string: " )
```

```
    minhaRegex = re.compile(r'\d{2} \d{4}(-)?\d{4}?')
```

```
    resultado = minhaRegex.search(texto)
```

```
    print(resultado.group())
```

Note que, como tem [grupos na Regex](#), usamos os [métodos compile](#) e o [search](#).

A única de diferente que fizemos foi colocar o hífen num grupo (parêntesis) e colocamos o ? depois dele, pra simbolizar que ele é opcional.

- **Asterisco em Regex: ***

O símbolo de asterisco, em expressões regulares em Python, quer dizer 'corresponde a zero ou mais ocorrências'. Ele vem depois de um grupo, e quer dizer:

'Ei, esse padrão aqui, pode ocorrer nenhuma vez, uma, duas, três...um milhão, tanto faz'

Por exemplo, vamos supor que queiramos detectar os seguintes padrões de números de telefone:

- xx yyyy-zzzz
- xxyyyy-zzzz
- xx yyyy-zzzz
- xx yyyy-zzzz

Ou seja, entre o DDD e o número pode ter nenhum espaço, um, dois, vários espaços (nunca duvide da capacidade do usuário de fazer me..sbla).

Tudo que temos que fazer é adicionar o grupo: **()***

Ele diz: "Python, pode ter um espaço vazio aí, nenhum, talvez dois...ou 1 bilhão, ok?"

Vejamos como fica nosso script:

```
import re
```

```
while True:
```

```
    texto = input("Digite sua string: " )
```

```
    minhaRegex = re.compile(r'\d{2}( )*\d{4}-\d{4}?')
```

```
    resultado = minhaRegex.search(texto)
```

```
    print(resultado.group())
```

- **Sinal de Adição em Regex Python: +**

O caractere de adição tem uma função bem parecida com o de asterisco.

A diferença é que, no asterisco, ele corresponde a **zero** ou mais ocorrências de um determinado grupo.

No símbolo de adição, ele corresponde a **uma** ou mais ocorrências daquele determinado grupo.

Ou seja, aquela ocorrência tem que ocorrer pelo menos uma vez.

Se você desejar que sua regex detectadora de número de telefone tenha **pelo menos um** espaço entre o DDD e o número, para detectar:

- xx yyyy-zzzz
- xx yyyy-zzzz
- xx yyyy-zzzz

Basta colocar + após grupo (). Veja como fica nosso código Python:

```
import re

while True:
    texto = input("Digite sua string: ")

    minhaRegex = re.compile(r'\d{2}(\ )+\d{4}-\d{4}?')
    resultado = minhaRegex.search(texto)
    print(resultado.group())
```

Classes de Caracteres

Neste tutorial de Python, sobre Strings e Regex, vamos estudar as principais classes de caracteres em expressões regulares.

- **Classes de Caracteres de Regex em Python**

Até o momento, em nossos tutoriais sobre Regex, usamos uma classe de caractere:

`\d` - os dígitos.

Todo e qualquer dígito, de 0 até 9, representamos por `\d`.
Ele é uma classe de caractere.

Porém, é apenas uma das classes, e usamos somente ela para fins didáticos.

Vamos ver agora as classes:

- `\d` - Qualquer dígito de 0 a 9
- `\D` - Qualquer caractere que não seja um dígito (contrário de `\d`)
- `\w` - Qualquer letra, dígito ou o caractere underscore (qualquer caractere de uma palavra)
- `\W` - Qualquer caractere que não seja uma letra, um dígito ou o underscore (contrário de `\w`)
- `\s` - Qualquer espaço, tabulação ou caractere de quebra de linha, ou seja, qualquer espaçamento
- `\S` - Qualquer caractere que não seja um espaço, uma tabulação ou uma quebra de linha (contrário de `\S`)

Vamos supor que queiramos detectar o seguinte padrão:

1. Um ou mais números
2. Um espaçamento
3. Um ou mais caractere

Para detectar um ou mais número: `\d+`

Para detectar apenas um espaçamento: `\s`

Para detectar um ou mais caractere, seja dígito ou não: `\w+`

Logo, nossa regex é: `\d+\s\w+`

Você pode usar essa expressão regular para 'catar' receitas pela internet, note que elas tem esse padrão:

1 cebola

2 bifes

3 alhos

Exemplos de Classes de Caracteres

Os 6 tipos de classes apresentadas, são os 'geralção', do Python.

Você também pode criar suas próprias.

Por exemplo, se desejar detectar dígitos de 0 até 5, use a Regex:

`r'[0-5]'`

Para detectar vogais minúsculas:

`r'[aeiou]'` ou `r'[a-u]'`

Para detectar tanto letras minúsculas como minúsculas:

`r'[a-zA-Z]'`

Para detectar um padrão formado só por letras do alfabeto e dígitos:

`r'[a-zA-Z0-9]'`

E para detectar o contrário do que você deseja, basta inserir o acento circunflexo antes `^`.

Por exemplo, detecta tudo menos letras do alfabeto e dígitos:

`r'^[a-zA-Z0-9]'`

Detectar tudo que não é dígito:

`r'^\d'` que é o mesmo de `r'\D'`

Início e Fim de String: Acento circunflexo (^) e Sinal de Dólar \$

- **Início de String - Acento circunflexo: ^**

Muitas vezes, desejamos detectar um padrão que esteja exatamente no início de uma string.

Um caractere especial, o de acento circunflexo, serve para simbolizar o início de uma string: ^

Ou seja, basta posicionar esse caractere em sua regex, e vai começar a trabalhar com o começo de qualquer string. O script abaixo detecta a palavra "Olá", no início de qualquer string:

```
import re

while True:
    texto = input("Digite sua string: " )

    minhaRegex = re.compile(r'^Olá')
    resultado = minhaRegex.search(texto)
    print(resultado.group())
```

O script Python abaixo, é mais generalista, ele vai te informar a primeira palavra da string:

```
import re

while True:
    texto = input("Digite sua string: " )

    minhaRegex = re.compile(r'^(\w)+\s')
    resultado = minhaRegex.search(texto)
    print(resultado.group())
```

O que fizemos foi, no início da string (^), detectar um ou mais (+) caractere de escrita (\w), até o primeiro espaçamento (\s).

- **Final de String - Símbolo de dólar: \$**

Analogamente ao início de string e circunflexo, muitas vezes é necessário detectar o final da string.

Esse fim é simbolizado pelo sinal de dinheiro, o dólar: \$

O script abaixo detecta se os quatro últimos caracteres de uma string são: ponto e três letras

Ou seja, ele detecta se você digitou o nome de um arquivo e sua extensão, como .txt ou .png ou .exe

```
import re
```

```
while True:
```

```
    texto = input("Digite sua string: " )
```

```
    minhaRegex = re.findall(r'\.\w{3}$', texto)
```

```
    print(minhaRegex)
```

Detectando ponto: \.

Detectando 3 caracteres: \w{3}

Detectando final da string: \$

Já o script abaixo detecta padrões compostos somente por dígitos, com um ou mais presente:

```
import re
```

```
while True:
```

```
    texto = input("Digite sua string: " )
```

```
    minhaRegex = re.compile(r'^(\d)+$')
```

```
    resultado = minhaRegex.search(texto)
```

```
    print(resultado.group())
```

Veja que o padrão é:

Início da string: ^

Um ou mais dígitos: \d+

Final de string: \$

Caractere Curinga - Ponto: .

- **Caractere de Ponto: .**

O caractere de ponto . é dito curinga, pois ele substitui todo e qualquer caractere, exceto a quebra de linha \n

O script abaixo, detecta toda e qualquer palavra terminada por 'ato':

```
import re
```

```
while True:
    texto = input("Digite sua string: ")

    minhaRegex = re.findall(r'.ato', texto)
    print(minhaRegex)
```

Veja um exemplo: 'o gato comeu um rato e um pato':

```
Digite sua string: O gato comeu o rato e o pato
['gato', 'rato', 'pato']
```

Ou seja, ele mostra as palavras que riam com *ato*.
Lembre-se que um ponto substitui apenas um caractere.

E que, para detectar um ponto numa string, você deve escapar o ponto: \.

- **Detectando tudo: .***

Muitas vezes, é necessário detectar *tudo*.

Por exemplo, você recebe em seu sistema, da empresa onde trabalha, uma string na forma de um formulário:

Nome: ...

Idade: ...

Função: ...

Se quisermos detectar tudo após **Nome:**, usamos os dois caracteres:

```
r'.*'
```

Lembre-se: o caractere curinga de ponto, detecta tudo, menos quebra de linha.

Ou seja, o uso desses dois caracteres `.*` vai pegar tudo até o final da linha, ou seja, todo o nome da pessoa.

Ou seja, é um dos padrões mais usados e conhecidos, em expressões regulares.

- **Método `re.DOTALL`**

Para fazer com que o caractere curinga, ponto `.`, reconheça a quebra de linha como um caractere também, basta passarmos o valor `re.DOTALL`, como segundo argumento da `re.compile()`

- **Maiúsculo e Minúsculo: `re.IGNORECASE` ou `re.I`**

Se desejarmos que a regex passada dê *match* sem se importar se as letras são maiúsculas ou minúsculas, basta passarmos `re.IGNORECASE` ou simplesmente `re.I` como segundo argumento para a `re.compile()`

Substituindo strings: Método `sub()`

- **Fazer Substituição em Regex: Método `sub()`**

Até o momento, em nossos tutoriais de regex, aprendemos a detectar determinados padrões em textos, usando expressão regular.

Porém, é comum que, ao invés de apenas detectar, queiramos *substituir* uma coisa por outra.

Por exemplo, se seu amigo diz:

- Funk é o melhor estilo musical

Seria legal ser substituído por:

- Rock é o melhor estilo musical

Isso pode ser feito através do método `sub()`, presentes em objetos Regex.

Esse método recebe dois argumentos:

A string, o texto ou valor que vamos querer colocar no lugar de outra

A string onde irá ocorrer a substituição

Ou seja: `minhaRegex.sub(substituirPorIsso, antigaString)`

O script abaixo substituir o padrão 'Funk' por 'Heavy Metal':

import re

while True:

 texto = input("Digite sua string: ")

 minhaRegex = re.compile(r'Funk')

 minhaRegex = minhaRegex.sub('Heavy Metal', texto)

print(minhaRegex)

Veja o resultado:

```
Digite sua string: Funk é o melhor estilo musical
Heavy Metal é o melhor estilo musical
Digite sua string:
```

- **Texto correspondente na substituição: \1, \2, \3...**

Muitas vezes, é comum que queiramos usar o texto correspondente (que vai ser encontrado pelo padrão), pra usar na própria substituição.

Por exemplo, o primeiro grupo da Regex, será simbolizado por \1.

O segundo grupo, por \2 .

E assim por diante.

Por exemplo, a string de regex: r'padrao1(padrao2)padrao3(padrao4)'

O *padrao2* pode ser simbolizado por \1

O *padrao4* pode ser simbolizado por \2

O script abaixo detecta se foi digitado *Funk*.

Se sim, essa correspondência vai ser armazenada em \1

Então, substituímos ela por '\1 (eca)'

Ou seja, onde tiver 'Funk', vai aparecer 'Funk (eca)':

```
import re
```

```
while True:
```

```
    texto = input("Digite sua string: " )
```

```
    minhaRegex = re.compile(r'(Funk)')
```

```
    minhaRegex = minhaRegex.sub('\1 (eca)', texto)
```

```
    print(minhaRegex
```

Criando Expressões Regulares Longas e Complexas

- **Modo verbose: `re.VERBOSE`**

Ao estudar expressões regulares, você foi, aos poucos, entendendo aquela sopa de letrinhas e passou a fazer o maior sentido para você.

Mas mostre aquilo para sua mãe ou amigos, vão achar que você ficou doido, está escrevendo em um dialeto alienígena e vão querer te internar.

E não é pra menos, a medida que nossas regex vão ficando mais longas e complexas, vai ficando muito mais difícil de entender elas, isso pois os símbolos ficam 'grudados' um nos outros.

Para facilitar e resolver isso, existe um argumento do método `re.compile` chamado `re.VERBOSE` que vai ignorar todo o espaços em brancos e comentários na string que define a regex.

Ou seja, basta fazer: `re.compile(r'padrao', re.VERBOSE)`

- **Detectando Números de telefone**

Agora vamos fazer o script definitivo, que encontra todo e qualquer número de telefone em um texto, site, arquivo extenso ou o que for.

Se tiver dois dígitos de DDD, 4 números e depois 4 números, ele detecta.

- Primeiro grupo

Primeiro, temos que detectar o DDD.

Ele pode aparecer de uma das seguintes maneiras:

xx

(xx)

Regex pro primeiro caso: `\d{2}`

Regex do segundo caso: `\(\d{2}\)`

Como pode ser um ou outro, colocamos um pipe `|` entre essas duas possibilidades.

- Segundo grupo

Aqui, depois do DDD, pode ter algum espaçamento ou não.
Representamos isso assim: `(\s*)?`

O asterisco é pra caso tenha várias espaços.
A interrogação é que esse grupo pode existir ou não.

- Terceiro grupo

Agora, vem os 4 primeiros dígitos do telefone em si:
`(\d{4})`

- Quatro grupo

Agora, pode ser que exista um hífen ou algum espaçamento.
Pode ter um ou mais espaçamento: `\s*`
Pode ter um hífen, vários, under line ou o que seja: `.*`

Como pode ser um ou outro, colocamos um pipe | entre eles

- Quinto grupo

Por fim, os quatro últimos dígitos:
`(\d{4})`

Nosso código Python fica:

```
import re
```

```
while True:
```

```
    texto = input("Digite sua string: " )
```

```
    minhaRegex = re.compile(r"(\d{2}|\d{2}\d)?          # código de área
                              (\s*)?                    # espaço
                              (\d{4})                    # primeiros 3 dígitos
                              (\s*|.*?)                  # separador
                              (\d{4})                    # últimos 4 dígitos
                              )", re.VERBOSE)
    minhaRegex = minhaRegex.search(texto)
```

```
print(minhaRegex.group())
```

Veja que colocamos entre aspas triplas, para poder quebrar em várias linhas. Também, fizemos um comentário sobre cada grupo, para facilitar o entendimento.

Ficou bacana, não ficou ?

- **Regex para detectar um endereço de e-mail**

Um endereço de e-mail é composto de:

1. Nome de usuário
2. Arroba
3. Nome do domínio
4. Ponto
5. Extensão do domínio (com, com.br, net, pt etc)

Nosso código fica:

```
emailRegex = re.compile(r'''([a-zA-Z0-9._%+-]+      # nome do usuário
@              # arroba
[a-zA-Z0-9.-]+  # domínio
(\.[a-zA-Z]{2,4}) # ponto seguido de outros caracteres
)''', re.VERBOSE)
```

Ou seja, o nome deve conter pelo menos um caractere (maiúsculo ou minúsculo, ou algum dígito), seguido de @, depois a mesma regex para nomes, um ponto mais 2 a 4 caracteres.

Orientação a Objetos

Nesta seção, vamos estudar um dos modelos de programação mais fantásticos e úteis existentes: a orientação a objetos.

No decorrer de nosso curso, falamos e usamos vários conceitos da *POO* (Programação Orientada a Objetos) como objetos, métodos, atributos etc.

Agora, vamos nos aprofundar nesse estudo e criar nossas próprias classes, objetos, métodos, atributos e começar a criar *scripts* e *softwares* maiores, mais robustos e realistas, fazendo programas bem parecidos com os que usamos no nosso cotidiano.

Vale lembrar que a POO em Python é totalmente opcional: usa quem quer, quando quiser e se quiser.

De fato, existe muita coisa boa feita com e sem POO.

A [linguagem Java](#), é OO, não tem como fugir da orientação a objetos.

A [linguagem C](#) é estruturada, não tem como trabalhar com POO em C (com C++ sim, mas é outra linguagem).

Já o nosso querido e amado Python, oferece ambas opções.

Foda esse Python, não é?

Classe e Objeto em Python - O que é ?

Classe é o principal componente ou dispositivo, da orientação a objetos. Imagine uma classe como uma planta (como a planta de uma casa). A planta é a classe que serve para criar as casas, que são os objetos. É como se a classe fosse um molde, um modelo, um rascunho de algo.

Por exemplo, o Carro é um exemplo de classe. Ele representa um objeto que tem pneu, motor, portas, volante, câmbio etc.

Porém, ninguém chega na concessionária e diz:

- Olá, quero um carro
- Ok, aqui está um carro, custa 50 mil.
- Obrigado, agora tenho um carro

Na verdade, você compra coisas específicas:

- Um celta, um Gol, um Corolla ou uma BMW (se você for [Programador Python Profissional](#)).

Ou seja, a classe Carro é uma abstração, um modelo. A coisa real, são os objetos, (Palio, Uno, Hilux etc).

Todo Gol tem características da classe Carro. Assim como todo Palio, Uno, Civic etc.

• Exemplo de Classe e Objeto

Um ser humano é uma Classe.

Vamos tentar definir características gerais de uma pessoa?

Ela tem:

- Cabeça
- Coração
- Cérebro
- Rosto
- etc

Mas você não lida no dia-a-dia com a Pessoa.

Você lida com o João, a Maria, ou seja, você lida com pessoas concretas e específicas, são objetos.

Objeto é algo concreto, que existe. A Classe é apenas uma 'planta', uma definição generalista.

O João é um objeto, ou instância, da classe Pessoa.

O Honda Fit é um objeto, ou instância, da classe Carro.

- **Orientação a Objetos**

As classes funcionam como verdadeiras fábricas de objetos (instâncias). Sua principal característica é ter dados específicos e internos, bem como métodos que irão trabalhar nos objetos e atributos próprios.

Todo objeto vai herdar os atributos e métodos da sua classe. Ou seja, todo objeto do tipo Carro, vai ter atributos (informações) sobre potência do motor, número de portas, velocidade máxima atingida etc.

Esses objetos também terão métodos pré-definidos na Classe, como o método que faz o carro ligar, o método que faz o motor girar, o método pra trocar de marcha etc etc etc.

Na verdade, durante todo nosso **curso de Python**, usamos vários objetos, de tipos internos, nativos da linguagem, como objetos do tipo Regex, do tipo Lista, string etc.

Agora, vamos poder criar nossos próprios 'tipos' de dados. Como o tipo Carro ou o tipo Pessoa ou qualquer outro objeto que queiramos, bastando pra isso definir bem nossas classes.

- **Vantagens da Orientação a Objetos**

Não importa qual paradigma você use, existe sempre uma porção de vantagens e desvantagens.

E tenha em mente uma coisa: não há melhor nem pior paradigma, assim como não existe linguagem melhor nem pior. O que existe é um paradigma, ou linguagem, mais apropriado para cada objetivo.

Vamos definir aqui algumas vantagens do uso da orientação a objetos.

- **Infinitos objetos**

Uma vez que você criou a classe Carro, você pode criar um objeto do tipo carro, como um Celta. Depois um Gol, depois...outro Gol ou outro Celta...ou 1 milhão de Palios.

Ou seja, bastou criar a classe Carro, depois você gera quantos objetos quiser. E todos terão as mesmas características gerais, todos serão carros (embora tenham alguns detalhes diferentes, como potência ou número de portas, mas todos tem alguma potência e um determinado número de portas).

- Herança

Dizemos que os objetos *herdam* características da Classe que os criou. Vamos supor que antigamente todos usavam gasolina.

Um país criou 10 milhões de objetos, ou seja, 10 milhões de carros diferentes (objetos).

Depois, passou a existir combustível do tipo álcool.

Logo, apareceu uma característica diferente: que combustível os carros usam.

O que fazer?

Simples, é só ir lá na Classe e criar mais um atributo: o tipo de combustível. Automaticamente todos os objetos desse tipo terão esse atributo, não vai precisar ir em um por um criando essa característica.

Tem esse atributo na Classe? Vai ter em todos os objetos, pois todo objeto é uma instância de alguma classe.

- Composição

Existem vários e vários tipos de objetos e classes, no mundo.

Tem a classe Moto, com suas características.

Tem a classe Pedestre, com seus atributos específicos.

Tem a classe Estrada, por exemplo: pode ser avenida, rua, ser asfaltada, de barro etc.

Esses objetos *lidam* entre si, existem em conjuntos. Quando um vai passar, o outro tem que parar (ou vai acontecer um acidente).

Se um passa num sentido da via, o outro tem que passar no outro sentido. Uma moto vai ultrapassar o carro? Ok, mas existem *regras* pra isso acontecer.

Resumindo: o mundo é formado por diversos componentes, diversos objetos coexistindo e se relacionando por meios específicos (*métodos*). Alguns objetos possuem outros e por aí vai.

Cada objeto *compõe* um cenário maior, para o mundo girar e seguir girando e funcionando.

- Encapsulamento

Uma das características mais fantásticas da POO, é o encapsulamento.

Cada classe *encapsula* atributos e métodos.

Atributos são informações, características.

Por exemplo, na classe Carro, alguns atributos: cor do carro, tamanho dos pneus, potência do motor, ano de fabricação etc.

Já os métodos são as *funcionalidades*: mecanismo que faz o motor dar a partida, mecanismo que refrigera o sistema, mecanismo que gera eletricidade.

Ou seja, dentro de uma classe (logo, dentro de todo objeto), existem informações e funções específicas, que existem só ali e somente para aquele tipo de objeto, que são encapsulados e *guardados* do mundo exterior.

Assim, você só acessa alguma informação de um objeto se criar um método que vai permitir isso, senão fica invisível pra todos os outros componentes do seu programa.

Mas esse papo já está ficando muito teórico, muito 'viagem'...nos próximos tutoriais vamos botar a mão na massa, criar e fazer as coisas acontecerem de verdade, com nossas próprias mãos...digo, com nossos próprios códigos Python.

Como Criar Classes e Objetos

Como definir e criar uma classe

Definimos uma classe através da instrução **class**, seguido de um nome para sua classe, dois pontos e as instruções:

```
class MinhaClasse:  
    [codigo da classe]  
    [ex:atributos]  
    [ex:métodos]
```

Prontinho. Você acabou criar uma classe de nome **MinhaClasse**. Abaixo dela, indentado, vai todo o seu código.

Nesse trecho do código onde vai os métodos que irão trabalhar e atuar nos objetos, bem como seus atributos (informações).

- Quer trabalhar como programador Python? [Obtenha seu certificado](#).

• Como Criar e Instanciar um Objeto

Agora que já criamos a classe **MinhaClasse**, vamos aprender como criar, ou melhor dizendo: como *instanciar* objetos dessa classe.

Primeiro definimos o nome da variável, depois fazemos ela receber o nome da classe, seguido de um par de parêntesis.

- meuObjeto1 = MinhaClasse()
- meuObjeto2 = MinhaClasse()
- etc

Prontinho. Só isso.

Todo o código definido para **MinhaClasse** será um código interno de cada objeto. Ou seja, todos os objetos que você instanciou (imagine a classe como uma fábrica de objetos, uma fabricante de instâncias de objetos) vão todos os métodos e atributos da **MinhaClasse**.

E não importa se você instanciou nenhum, um ou 1 bilhão de objetos. Todos vão *herdar* essas informações automaticamente.

Ah...e mudou o código da sua classe?
Passa a valer pra todas as instâncias.

- **Exemplo de Classe e Objeto em Python**

Copie o código abaixo e rode ele:

```
class Carro:
    def __init__(self):
        print("Carro criado")
```

```
corolla = Carro()
civic = Carro()
```

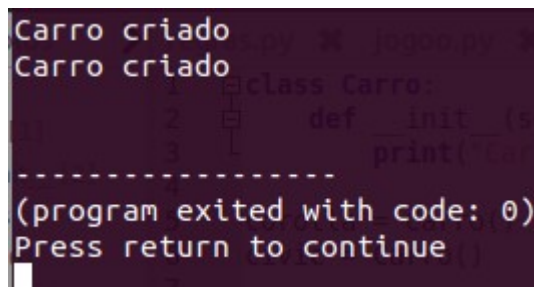
Definimos a classe Carro nele.

Escrevemos um método nele, o `__init__`, que recebe o parâmetro *self* (são dois *underline* antes da palavra **init** e dois depois, ok?)

E método é o mesmo que [função](#), mas função interna de uma classe, que só existe e atua nos seus objetos. Ela só é vista fora do escopo do objeto se o programador desejar. Veremos essa questão de *segurança* depois também.

Esse método, que iremos explicar melhor no próximo tutorial o que é e para que serve, manda a mensagem na tela: *Carro criado!*

Instanciamos dois objetos dessa classe, o *corolla* e o *civic*, veja o resultado:



```
Carro criado
Carro criado
-----
(program exited with code: 0)
Press return to continue
```

Ou seja, você acabou de criar uma classe e objetos, já está vendo a programação orientada a objetos em funcionamento! Show!

Mas a pergunta que não quer calar: qual melhor, Corolla ou Civic ?

Métodos, Método `__init__` e Atributos

- Método `__init__`

A regra para se criar uma classe é simples e você já conhece, se cria do mesmo jeito que se cria uma [função em Python](#): instrução `def`, nome da função, parêntesis (com parâmetros ou não), dois pontos e o código que vai ser executado.

No tutorial passado, onde ensinamos [como criar classes e objetos](#), usamos um método muuuuito especial, o `__init__`

Esse método é especial porque é, geralmente, o primeiro definido em toda classe.

O motivo de ser especial é que ele sempre é executado quando criamos uma instância de um objeto.

Automaticamente o Python invoca o `__init__()` quando você cria um objeto.

- O parâmetro *self*

Como explicamos, o código da classe é apenas um **molde** dos objetos, uma forma.

Não existem especificamente uma classe. Um objeto sim, existe especificamente.

E é em cada objeto, específica e unicamente, que os métodos vão atuar, e é por isso que o método **self** é necessário em cada método de uma classe, pra simbolizar que estamos operando em uma informação de objeto, especificamente.

self, em inglês, quer dizer auto, própria, ele mesmo.

Ou seja, se refere aquele objeto, em específico.

Pode criar 1 trilhão de objetos de uma mesma classe. Mas o **self** se refere aquele objeto específico, ok ? Quando um método executa, ele precisa saber em qual destes objetos existentes ele vai precisar atuar.

- **Como Criar e Acessar Atributos**

Ok, já vimos um método em ação em POO, o `__init__` (que como veremos mais a frente, é um método construtor).

Mas as classes também encapsulam outra coisa importante além de métodos, os **atributos**, que nada mais são que informações, variáveis onde iremos armazenar dados.

Por exemplo, vamos criar o atributo **portas** e inicializar com o valor 2. Se criamos um objeto de nome **Celta** e ele tem um atributo chamado **portas**, acessamos esse atributo através de ponto, assim:

- Celta.portas

Se tem um atributo chamado **motor** num classe de onde instanciamos o objeto BMW, acessamos os dados desse motor assim:

- BMW.motor

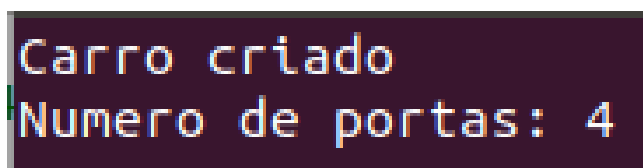
Vejamos um exemplo:

```
class Carro:
    portas = 4

    def __init__(self):
        print("Carro criado")

corolla = Carro()
print("Numero de portas:", corolla.portas)
```

Rodando, o resultado vai ser:



```
Carro criado
Numero de portas: 4
```

Bem simples e intuitivo, não?

Se puxar pela memória, já usamos o atributo **pi** da **math**, em nossos exercícios da seção básica: *math.pi*, embora seja um pouco diferente pois *math* é um módulo, mostra a lógica de usar o ponto.

`math.pi` = quero usar a variável **pi** que está em **math**

`corolla.portas` = quero usar a variável **portas** que está no objeto **corolla**

- **Como Criar um Método**

Aprendemos que a função interna de uma classe, chamada `__init__` é um método, que por definição, é executada sempre que instanciamos um objeto.

Porém, é possível também criar nossos próprios métodos.

A recomendação é que você crie métodos para trabalhar com os dados da própria Classe, eles servem para moldar e definir o *comportamento* dos objetos.

Por exemplo, vamos criar o método **exibePortas()**

Ela simplesmente retorna o valor de portas do carros, veja como fica nosso código:

```
class Carro:
    portas = 3
    def __init__(self):
        print("Carro criado")

    def exibePortas(self):
        return self.portas
```

```
veloster = Carro()
print("Numero de portas:", veloster.exibePortas())
```

Note que o primeiro parâmetro de todo método, é **self**.

Se quiser retornar um atributo de uma classe, também use o **self**:

```
self.portas
```

Sempre, ok?

E para invocar o método **exibePortas()**, do objeto **veloster**, use o ponto também:

```
veloster.exibePortas()
```

Embora esse método receber o parâmetro **self**, você não precisa enviar nenhum argumento para este parâmetro.

No próximo tutorial vamos aprender como usar mais parâmetros e argumentos, para criar métodos mais interessantes.

Parâmetros e Argumentos em Métodos

- **Método em Python**

No tutorial anterior, de nosso **curso de Python**, vimos como usar métodos em nossas classes.

Além de conhecermos o método construtor `__init__`, criamos nosso próprio método, o **exibePortas**.

Vamos agora ver um método mais interessante.

A classe inicia com o atributo **portas** com valor 2, por padrão.

Mas isso pode ser mudado, e vai ser possível fazer isso no método **definePortas**.

Ele pede ao usuário um número de portas, que seu carro deve ter e armazena na variável *num*.

Em seguida, ele atribui faz com que o atributo **portas** receba esse valor.

Veja que o atributo deve ser usado com *self*: **self.portas**

Veja como fica nosso código:

```
class Carro:
    portas = 2
    def __init__(self):
        print("Carro criado")

    def definePortas(self):
        num=int(input("Numero de portas:"))
        self.portas = num

    def exibePortas(self):
        return self.portas

carro = Carro()
carro.definePortas()
print("Numero de portas:", carro.exibePortas())
```

O código acima pede um número de portas ao usuário, atribui no atributo *portas* do objeto e exibe esse valor, via método **exibePortas**.

- **Argumentos e Parâmetros em Python**

A coisa que você mais deve se lembrar, ao trabalhar com métodos e atributos de uma classe, é de usar o **self**.

Agora vamos fazer com que a **definePortas** receba um valor número, para atribuir ao atributo de número de portas.

Nosso método vai ter dois parâmetros, o **self** (que deve sempre estar lá) e o valor que vamos receber, o **num**, veja como fica nosso método:

```
def definePortas(self, num):  
    self.portas = num
```

Agora, para chamar, você vai passar somente um argumento, o **num**, não precisa passar o **self**.

Veja bem: o método recebe dois parâmetros (self e um número), mas passamos apenas um argumento, o número de portas:

```
class Carro:  
    portas = 2  
    def __init__(self):  
        print("Carro criado")  
  
    def definePortas(self, num):  
        self.portas = num  
  
    def exibePortas(self):  
        return self.portas
```

```
carro = Carro()  
num=int(input("Numero de portas:"))  
  
carro.definePortas(num)  
print("Numero de portas:", carro.exibePortas())
```

Bem simples e intuitivo, não?

- **Segurança em Programas Python**

De cara, pode parecer meio **bobo** ter dois métodos: um para definir o valor do número de portas, e outro para exibir esse valor.

Mas esse é o padrão: o mundo externo (programa fora da classe), deve ter o poder de mexer nos objetos através apenas de métodos específicos.

Não é seguro, em **hipótese alguma**, que seja possível ter acesso ao atributo **portas** senão via esses métodos. Ou seja, nenhum bug ou hacker deve ter acesso a esse atributo, somente por algum método.

Método este que deve 'tomar as rédeas' dos atributos.

No exemplo abaixo, vamos criar uma classe que representa a conta do banco de um cliente.

Temos a variável saldo, que inicia com valor R\$ 0.00

Temos o método **exibeSaldo**, que retorna o saldo da pessoa (objeto) e também o método **depositae**.

Esse método de depositar, só deve funcionar se o valor depositado seja positivo.

Caso contrário, não é possível realizar depósito. Pode reclamar, tentar invadir, hackear ou o que for, mas só vai depositar um valor positivo:

```
class Conta:
    saldo = 0.0
    def __init__(self):
        print("Conta do cliente criada")

    def depositar(self, num):
        if(num>0):
            self.saldo = num
        else:
            print("Nao é possível depositar 0 ou menos")

    def exibeSaldo(self):
        return self.saldo

pessoa = Conta()
print("Saldo inicial:" , pessoa.saldo())
num=int(input("Depositar:"))

pessoa.depositar(num)

print("Saldo atual:" , pessoa.saldo())
```

- **Exercício de Orientação a Objetos**

Agora crie um método de **saque**, onde só deve ser possível fazer um saque se a pessoa tiver um saldo positivo, e o saque máximo é esse valor.

O programa deve exibir um looping infinito, perguntando se a pessoa deseja depositar ou sacar valores.

Primeiro, definimos o saldo na conta do cliente como **saldo=0.0** afinal, o cliente acabou de criar a conta, não tem nada.

Para depositar, primeiro verificamos que o valor desejado para depósito seja maior que 0.

Se sim, ok, depositamos. Se não for, informamos o erro.

Depois, é o método **sacar**, que vai sacar um valor **valor**.

Esse valor deve ser menor ou igual ao saldo, pra poder sacar.

Se não der, exibe mensagem de erro.

Nosso código fica:

```
class Conta:
    saldo = 0.0
    def __init__(self):
        print("Conta do cliente criada")

    def depositar(self, num):
        if(num>0):
            self.saldo += num
        else:
            print("Nao é possível depositar 0 ou menos")
    def sacar(self, valor):
        if(valor<=self.saldo):
            self.saldo -= valor
        else:
            print("Valor insuficiente. Você tem:", self.saldo)

    def exibeSaldo(self):
        return self.saldo
```

```
pessoa = Conta()
```

```
op = True
```

```
while op:
    print("0. Sair")
    print("1. Exibir saldo")
    print("2. Depositar")
    print("3. Sacar")
    op=int(input())

    if(op==1):
        print("Saldo:", pessoa.exibeSaldo() )
    elif(op==2):
        num=float(input("Valor para depositar:"))
        pessoa.depositar(num)
    elif(op==3):
        num=float(input("Valor para sacar:"))
        pessoa.sacar(num)
    else:
        print("Saindo do sistema...")
```

Prontinho, você acabou de criar um sistema bancário em Python, onde um usuário pode fazer depósito, sacar e consultar seu saldo.

Simples, esse Python, não?

Método `__init__`: Parâmetros e Argumentos

- **Método Construtor:** `__init__`

O método `init` é bem especial, pois como já explicamos, ele é automaticamente executado sempre que instanciamos um objeto de uma classe.

Ele é chamado de **construtor**, e vimos ele apenas usando o parâmetro **self**:

```
def __init__(self)
```

Porém, é mais comum usarmos ele com outros parâmetros.

Por exemplo, para ele receber também um número, que representa o número de portas de um carro, através da variável *num*, fazemos:

```
def __init__(self, num)
```

E para usar esse método, basta passarmos um argumento ao criamos o Objeto.

Lembra que criamos um objeto da classe `Carro` da seguinte maneira:

```
palio = Carro()
```

Para criar esse mesmo objeto, já enviando o número de portas (2 por exemplo), fazemos:

```
palio = Carro(2)
```

Veja um exemplo de código:

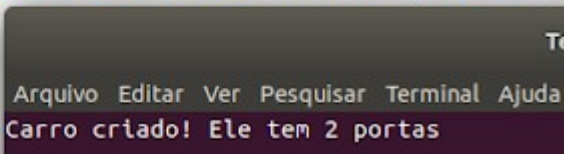
```
class Carro:
    def __init__(self, num):
        self.portas = num
        print("Carro criado! Ele tem", self.portas, "portas")
```

```
palio = Carro(2)
```

E o resultado:


```
class Carro:
    def __init__(self, num):
        self.portas = num
        print("Carro criado! Ele tem", self.portas, "portas")

palio = Carro(2)
```



Nesse código nosso método construtor já atribui o número de portas recebido a variável **portas**, interna de toda classe Carro. Veja que o método **init** tem dois parâmetros: **self** e **num**

Mas na hora de passar os argumentos, só passamos o *num*, pois já estamos indicando o objeto (*palio*) que estamos fazendo isso.

- **Passando Vários Argumentos**

Agora vamos fazer a mesma coisa, porém vamos usar dois parâmetros e dois argumentos.

Além do número de portas, primeiro vamos passar uma string, com o modelo do carro.

Vamos armazenar essa informação na variável **nome** (logo, **self.nome**), veja:

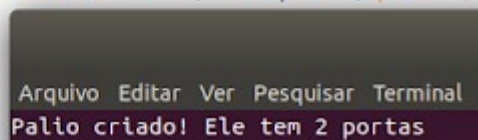
```
class Carro:
    def __init__(self, nome, num):
        self.nome = nome
        self.portas = num
        print(self.nome, "criado! Ele tem", self.portas, "portas")
```

```
palio = Carro("Palio", 2)
```

Resultado:

```
class Carro:
    def __init__(self, nome, num):
        self.nome = nome
        self.portas = num
        print(self.nome, "criado! Ele tem", self.portas, "portas")

palio = Carro("Palio", 2)
```



Note que fizemos:
self.nome = nome

Esse primeiro *nome*, junto com *self*, se refere ao atributo.
Já o segundo *nome*, se refere ao argumento, ok?

É como fazer:
`count = count + 1`

Usamos novamente a mesma variável, duas vezes, na mesma expressão.
Leia assim: *count* vai receber seu valor antigo *count* somado mais um.

- **Método Construtor**

Os métodos construtores são úteis para, como o nome diz, construir um objeto.

Por exemplo, se você cuida do sistema de uma empresa e tem a classe *Funcionario*.

Então, um novo empregado foi contratado e você vai cadastrá-lo no sistema, para isso, vai criar um novo objeto, o **funcionario[2112]**, ou seja, mais um objeto do tipo *Funcionario* na sua lista de empregados da firma.

Ora, todo empregado tem um nome, número, cargo e salário.

Já crie seu objeto com essas informações. O método construtor `__init__` serve justamente pra isso, pra criar logo um objeto fazendo o necessário e obrigatório:

- `funcionario[2112] = Funcionario("Jose da Silva", 2112, "Gerente", 4800.00)`

Prontinho, o Zé da Silva, de número 2112, vai ser gerente e ter um salário R\$ 4.800,00 (ele é gerente da equipe de programadores, obviamente estudou Python pra ganhar isso).

Cara ou Coroa e Jogando dados com Orientação a Objetos

- **Cara ou Coroa com Orientação a Objetos**

Primeiro, importamos a biblioteca **random**, para [gerar números aleatórios em Python](#).

Nossa classe vai se chamar **CaraCoroa** e vamos armazenar o lado da moeda na variável *lado*.

No método construtor `__init__` inicializamos essa variável como 'Cara', como se nossa moeda estivesse inicialmente virada com a Cara pra cima.

Depois, definimos o método `lançar()`, que vai **lançar** a moeda.

Vamos gerar números aleatórios de 0 até 1. Ou seja, pode sair 0 ou 1 quando usamos:

```
random.randint(0,1)
```

Se sair 0, o resto da divisão por 2 dá 0, cai no IF e setamos o lado da moeda como 'Cara'.

Se der 1, o resto da divisão por 2 dá 1, cai no ELSE e setamos o lado da moeda como 'Coroa'.

No final, retornamos essa variável.

Prontinho, agora é só criar nosso objeto **moeda**, do tipo **CaraCoroa**.

Perguntamos ao usuário se ele deseja sair ou lançar mais uma vez, e cada vez que ele quiser lançar a moeda, basta acessar o método **lança()** do objeto **moeda** que ele vai retornar Cara ou Coroa.

Código Python:

```
import random
```

```
class CaraCoroa:
```

```
    def __init__(self):  
        self.lado = 'Cara'
```

```
    def lançar(self):  
        if (random.randint(0,1))%2==0:  
            self.lado = 'Cara'
```

```
    else:
        self.lado = 'Coroa'
    return self.lado
```

```
moeda = CaraCoroa()
op=1
```

```
while op:
    op=int(input("0.Sair\n"
                "1.Jogar de novo\n"))
```

```
    if op:
        print(moeda.lancar())
```

- **Jogando dados com Programação Orientada a Objetos em Python**

Vamos agora criar a classe **Dado**.

Ela vai ser bem semelhante a classe CaraCoroa, diferença é que vamos trabalhar com números de 1 até 6, ao invés de duas strings ('Cara' e 'Coroa').

Inicializamos o lado do dado como 1.

No método lancar, simplesmente retornamos um número de 1 até 6:
return random.randint(1,6)

Criamos nosso objeto **dado** do tipo Dado.

E prontinho.

Ajeitamos nosso menu, para que você possa lançar tanto dados como moedas, e veja como ficou nosso script em Python:

```
import random
```

```
class CaraCoroa:
    def __init__(self):
        self.lado = 'Cara'

    def lancar(self):
        if (random.randint(0,1))%2==0:
            self.lado = 'Cara'
        else:
            self.lado = 'Coroa'
```

```
    return self.lado
```

```
class Dado:  
    def __init__(self):  
        self.lado = 1  
    def lancar(self):  
        return random.randint(1,6)
```

```
moeda = CaraCoroa()  
dado = Dado()  
op=1
```

```
while op:  
    op=int(input("0.Sair\n"  
                "1.Lançar Moeda\n"  
                "2.Lançar dado\n"))  
  
    if op==1:  
        print(moeda.lancar())  
    elif op==2:  
        print(dado.lancar())
```

Sempre que faltar uma moeda ou um dado, você pode usar esse script :)

Atributos Privados - Escondendo Informações

- **Dados públicos e privados**

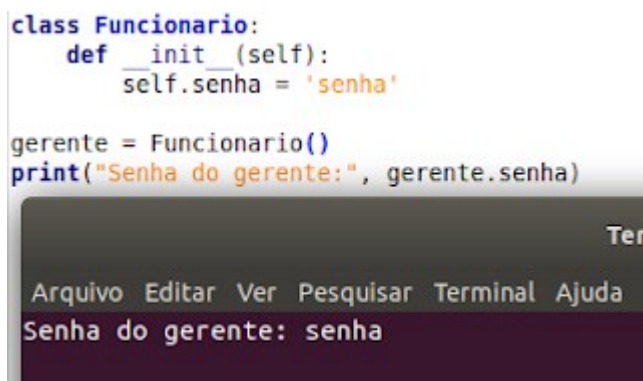
Vamos supor que você fez um sistema enorme, para uma multinacional. Dentro da classe **Funcionario** tem um atributo chamado *senha*, com a senha de cada funcionário, para eles fazerem o login no sistema da empresa.

Porém, um hacker invadiu seu sistema: deu um **print** nesse atributo:

```
class Funcionario:
    def __init__(self):
        self.senha = 'senha'

gerente = Funcionario()
print("Senha do gerente:", gerente.senha)
```

Com isso, é possível ver que a senha do usuário é *senha* (assuma, você já usou essa senha super criativa).



```
class Funcionario:
    def __init__(self):
        self.senha = 'senha'

gerente = Funcionario()
print("Senha do gerente:", gerente.senha)
```

Arquivo Editar Ver Pesquisar Terminal Ajuda
Senha do gerente: senha

Ou seja, temos aí uma falha! O ideal seria que algumas informações de nossas classes e objetos não fossem públicas assim

Vamos consertar isso.

- **Atributo Privado: __atributo**

Para tornar um atributo o mais privado possível, você deve adicionar antes do seu nome dois caracteres de *underline*.

Ou seja, se antes usávamos *senha*, agora vamos usar __senha

Veja o código e o que acontece quando tentamos acessar o atributo **gerente.__senha** agora:

```
class Funcionario:
    def __init__(self):
        self.__senha = 'senha'

gerente = Funcionario()
print("Senha do gerente:", gerente.__senha)
```

Terminal

Arquivo Editar Ver Pesquisar Terminal Ajuda

Traceback (most recent call last):

File "poo.py", line 6, in <module>

print("Senha do gerente:", gerente.__senha)

AttributeError: 'Funcionario' object has no attribute '__senha'

Isso mesmo, vai dar um erro!

O Python diz pro hacker que o atributo **__senha** não existe no objeto!

- **Segurança em Python**

É sempre bom definirmos variáveis privadas com dois *underscores*, pois isso mostra que aquele atributo é especial, ele não pode ser tratado por algo fora do objeto.

Ele deve ser acessado e modificado somente por métodos do próprio objeto. Por exemplo, um funcionário não precisa e nem deve ter acesso ao atributo de salário de outros funcionários, isso é uma informação particular.

Quem deve acessar esse atributo, por exemplo, é o método que calcula o imposto de renda no contra-cheque, ou o que desconta o valor do plano de saúde, pois essas coisas são baseadas no salário.

Mais na frente, durante nosso **curso de Python**, vamos falar mais sobre segurança, hábitos e procedimentos que devemos tomar para fazer um código mais seguro, robusto e estável.

Isso vai fazer de você um programador bem diferenciado no mercado.

Atributos de Classe

- **Atributos Gerais de Classe**

Como bem explanado e repetido em nossa seção de [Orientação a Objetos em Python](#), uma das grandes vantagens de usar este paradigma é a possibilidade de escrever o código apenas uma vez (ao definir a classe) e depois simplesmente sair instanciando (ou seja, criando) objetos.

Podemos criar milhões de objetos, fácil e rapidamente.
E cada um deles terá seus métodos e atributos próprios e únicos.

Porém, muitas vezes é interessante ter informações *gerais*, ou seja, um mesmo dado, um mesmo atributo presente em todos os objetos.

Veja bem: uma única variável, presente e passível de ser acessada e modificada por todos os objetos. Não é cada objeto ter sua variável, isso já sabemos como fazer.

Atributos de classes são aquelas variáveis que são únicas, só tem ela, e ela está presente em todos os objetos.

- **Como usar Atributos de Classe**

Vamos usar a variável **count** na classe **Funcionario**.

Ela é definida na classe inicialmente com valor 0, pois ainda não foi contratado nenhum funcionário.

Vamos armazenar os objetos (funcionários), numa lista **func**.

A ideia é: sempre que instanciarmos um funcionário, somamos 1 ao contador **count**.

Para fazermos isso no atributo de classe (e não no atributo de cada objeto), **não usamos self**.

Usamos a variável: *Funcionario.count*

Ao usar assim, todo objeto vai agir e somar 1, quando for criado, nesse atributo geral, de classe.

Veja bem:

- self.count -> vai atuar em um objeto, especificamente
- Funcionario.count -> todos vão atuar numa mesma variável, o atributo de classe

Veja nosso código onde você pode ir contratando quantos funcionários quiser.

Note que sempre que criamos um funcionário, incrementamos o contador e mostramos o total de funcionários existentes na empresa:

```
class Funcionario:
    count = 0
    def __init__(self, nome):
        self.__nome = nome
        print(nome, "contratado!")
        Funcionario.count += 1
        print("Numero de funcionarios:", Funcionario.count)

op=1
func = []
while op:
    op=int(input("0. Sair\n1. Criar funcionario\n"))

    if op==1:
        nome = input("Nome: ")
        func.append( Funcionario(nome) )
```

Vale ressaltar que é possível acessar o atributo **count** mesmo que você não tenha instanciado objeto algum, ok?

Composição: Classe usando Classe

- **O que é Composição em Python**

Até o momento, nos exemplos em que simulamos uma empresa, usamos apenas um tipo de objeto, da classe **Funcionario**, que representa os trabalhadores da empresa que você foi contratado para trabalhar.

Porém, existem uma porção de outros objetos nesta empresa. Aliás, vamos chamar a empresa de uma classe, a **Empresa**.

O que tem dentro dela?

Setores: RH, TI, Secretaria etc

Pessoas: Funcionários, Terceirizados etc

Local físico: Escritórios, Banheiros, Copa etc

Material: Móveis, Armários, folhas, canetas etc

Note uma coisa dentro dela: Funcionários.

Sim, *Empresa* é uma classe, e tem dentro dela outra classe, a *Funcionario*.

Podemos tratar o setor de RH e TI como classes também, e todos os demais exemplos acima.

Sua empresa é um objeto de classe *Empresa* que trabalha junto com o objeto *Correios*, por exemplo.

Podemos criar a classe *Almoxarifado* para lidar com material, onde por exemplo, cartucho de tinta ou caneta seriam objetos dessa classe.

Ou seja, na verdade, o nosso mundo é um punhado de objetos, **compostos** de outros objetos dentro, além de lidar com outros objetos de fora.

Damos a isso o nome de **composição**: combinação de objetos, quando instanciamos objetos de uma classe dentro de outra, quando usamos objetos de uma classe dentro de outros objetos.

Vamos ver um exemplo real de situação e código, para entendermos melhor o que é composição na prática.

- **Exemplo de Composição em Python**

Vamos usar novamente nossa classe **Funcionario**.

Para contratar um funcionário, você deve instanciar um objeto e passar uma string com o nome dele, e o método `__init__` já cria o objeto com o nome do funcionário.

Vamos criar outro método, o **retornaNome**, que retorna o nome do funcionário.

Agora vamos abrir uma empresa!

Vamos criar a classe **Empresa** que vai representar nossa firma em funcionamento.

Ela vai ter uma lista de funcionários, a **func**. Inicializamos essa variável como uma lista vazia, e a medida que formos contratando, vamos adicionando funcionários nele. E quem são esses? Objetos da classe *Funcionario*, ou seja, vamos estar fazendo composição!

Assim que o objeto da classe *Empresa* é criado, o método `__init__` é executado.

Dentro dele vamos colocar um looping infinito (*while True*), afinal, nossa empresa está sempre em funcionamento.

Perguntamos se o usuário deseja contratar alguém ou apenas ver a lista de funcionários.

Se ele desejar contratar, invocamos o método **contratar()**, que vai pedir um nome (variável *nome*) do funcionário, criar um objeto da classe *Funcionario*: `Funcionario(nome)` e vai colocar esse funcionário na nossa lista:

```
func.append( Funcionario(nome) )
```

Prontinho, funcionário contratador.

Se quiser ver todos os funcionários da sua empresa, basta digitar 2, que vai chamar o método **exibir()**, que simplesmente vai pegar a lista **func** de funcionários, ir em cada objeto dessa lista e executar o método *retornaNome()*, que vai exibir o nome de cada funcionário de sua empresa.

Prontinho. Agora, para criar nossa empresa, basta instanciar um objeto dela e o `__init__` já vai fazer tudo:

Empresa()

Veja como ficou nosso código Python:

```
class Funcionario:
    def __init__(self, nome):
        self.__nome = nome
    def retornaNome(self):
        return self.__nome

class Empresa:
    func = []
    def __init__(self):
        print("Empresa Tabajara em funcionamento")

        while True:
            print("1. Contratar")
            print("2. Exibir lista de funcionarios")
            op=int( input() )

            if op==1:
                self.contratar()
            elif op==2:
                self.exibir()
            else:
                print("Opção invalida")

    def contratar(self):
        nome = input("Nome: ")
        self.func.append( Funcionario(nome) )

    def exibir(self):
        for funcionario in self.func:
            print(funcionario.retornaNome())
```

Empresa()

Herança

- **O que é Herança em Python**

No sentido comum, o que é herança?

"Fulano morreu e deixou uma herança"

Ou seja, é quando alguém deixa algo para outro, algum patrimônio geralmente.

Em linguagem de programação Python, herança é uma espécie reusabilidade de código, onde uma classe é criada *herdando* atributos e comportamentos (métodos) de outra(s) classe(s).

Quando vamos criar uma classe (no dia-a-dia dos programadores Python), raramente criamos código 100% novo e original, o mais comum é usarmos *herança* para pegar algum código, ideia ou algoritmo já existente. Ou seja, não tentamos reinventar a roda, ela já existe, é melhor usar o que já tem de pronto.

Por exemplo, se quiser criar seu próprio conceito de Lista ou Dicionário, pode criar, nada te impede. Mas já existem classes criadas há anos, testadas por milhões de pessoas e *softwares*. Ora, é melhor usar o que já existe e já foi testado! Isso é Herança.

- **Exemplo de Herança em Python**

Vamos supor que criamos a classe **Veiculo**, ela representa todo e qualquer tipo de veículo motorizado, como carro, moto, caminhão, quadriciclo etc.

O que todo veículo tem em comum? Vamos criar alguns atributos:
Rodas, assentos, motor, potência, amortecedor etc.

Depois, criamos a classe **Carro** e vamos fazer com que ela *herde* os atributos da classe **Veiculo**.

Assim, automaticamente os atributos: rodas, assentos, motor, potencia etc.

Agora vamos criar a classe **Moto**, ela também vai herdar a classe **Veiculo**, logo ela também vai ter atributos: roda, assento, motor, amortecedor etc.

Ah, mas vai vao ser duas classes iguais!

Não, caro leitor.

Vamos herdar apenas as características comuns, que todo veículo tem. A classe *Carro* vai ter atributos e comportamentos especiais, como *ar-condicionado*.

Já a classe *Moto* não tem ar-condicionado.

Ou seja, herdamos o que tem de comum. O que tem de específico, aí sim programamos nessas classes *derivadas*.

Podemos ir mais além e criar classe **CarroPopular** que vai herdar os atributos e comportamentos da classe **Carro**, e por consequência, herda o comportamento e informações da classe **Veículo**.

Já a classe **SUV** também herda dados da **Carro**, porém tem suas diferenças da classe **CarroPopular**.

Estão captando a ideia?

Toda SUV é também um carro, logo também é um veículo. Mas tem coisas diferentes de um carro popular, embora os populares e SUV's sejam carros do mesmo jeito.

Ou seja, podemos dizer que no mundo real, existem classes mais *especializadas*, específicas de algum contexto geral. Uma moto é um veículo mais específico que simplesmente "Um veículo". Um Celta é algo mais específico que "Um carro". Mas a moto continua sendo um veículo e um celta continua sendo um carro.

A moto **é** um veículo.

Um Celta **é** um carro. Um carro **é** um veículo, logo, um Celta **é** um veículo.

No próximo tutorial vamos aplicar, no código, a Herança de classes, e vamos entender melhor esse importante conceito em programação orientada a objetos.

Como Usar Herança

-

Como fazer Herança

Vamos supor que temos a classe **Veiculo**:

```
class Veiculo:  
    #Métodos e atributos  
    #da classe veículo
```

Agora, desejamos criar a classe **Carro**, que vai herdar a **Veiculo**, basta colocarmos (Veiculos) entre parêntesis na definição de Carro:

```
class Carro(Veiculo):  
    #Essa classe vai herdar  
    #coisas da classe Veiculo
```

Ou seja, sempre que a classe **X** for herdar a classe **Y**, fazemos:
class X(Y):

E prontinho. O que tiver na classe **Y** (métodos e atributos), vai pra classe **X** também, sem precisarmos reescrever códigos e mais códigos. Agora, podemos escrever só os códigos específicos da classe **X** que não tem na **Y**,

- **Exemplo de Herança em Python**

Vamos lá, criar algum código e ver a herança funcionando na prática.

Primeiro vamos criar a classe **Veiculo**, que tem os atributos *tipo* (carro, moto, suv etc), *modelo* (celta, palio, honda, biz etc) e *km* que indica a quilometragem do veículo.

Nossa classe fica:

```
class Veiculo:  
    def __init__(self, tipo, modelo, km):  
        self.tipo = tipo  
        self.modelo = modelo  
        self.km = km
```

Agora vamos a classe **Carro**, que vai herdar a classe *Veiculo*. Ela recebe os atributos *tipo*, *modelo*, *km* e o *portas*.

Como os três primeiros são da classe base, a *Veiculo*, basta chamarmos o método `__init__` da classe *Veiculo* passando tais argumentos.

O único parâmetro novo é o *portas*, pois *Carro* tem porta, mas nem todo *Veiculo* tem porta.

Criamos também um novo método, o **exibe()**, que vai exibir os dados do carro. Veja como ficou nossa classe.

```
class Carro (Veiculo):
    def __init__(self, tipo, modelo, km, portas):
        Veiculo.__init__(self, tipo, modelo, km)
        self.portas = portas

    def exibe(self):
        print(self.tipo, "modelo", self.modelo, "com", self.km,
              "km rodados e", self.portas, "portas.")
```

Nosso código completo, criando um objeto do tipo *Carro*, que é um *Palio*, de 2 portas e 10.000km rodados:

```
class Veiculo:
    def __init__(self, tipo, modelo, km):
        self.tipo = tipo
        self.modelo = modelo
        self.km = km

class Carro (Veiculo):
    def __init__(self, tipo, modelo, km, portas):
        Veiculo.__init__(self, tipo, modelo, km)
        self.portas = portas

    def exibe(self):
        print(self.tipo, "modelo", self.modelo, "com", self.km,
              "km rodados e", self.portas, "portas.")
```

```
palio = Carro("Carro", "Palio", "10000", 2)
palio.exibe()
```

Rode e veja o resultado do código acima.

Note que não declaramos na classe *Carro* os atributos *tipo*, *modelo* e *km*. Eles foram herdados automaticamente da classe *Veiculo*.

Fizemos: *self.portas = portas* pois esse atributo é novo, existe somente na classe **Carro** e não tem na **Veiculo**. Captou a ideia da herança?

Nomenclatura importante:

Veiculo é chamada de superclasse.

Carro é chamada de subclasse.

Note que invocamos o método `__init__` da superclasse, para ela iniciar os parâmetros. Fazendo isso, automaticamente inicializa os atributos da subclasse, pois um objeto do tipo *Carro* é um objeto do tipo *Veiculo* também.

Tem algo na superclasse? Vai ter na subclasse também, seja método ou atributo.

- **Exercício de Herança**

Adicione uma classe ao script, a *Moto*.

Ela deve herdar a superclasse **Veiculo** e deve ter um atributo novo, o *cilindrada* e ter um método *exibe()* que printa todas as suas características.

Crie um objeto da subclasse *Moto* e chame o método *exibe()* para ver se funcionou.

- **Sistema bancário com Herança em Python**

Vamos agora criar um pequeno sistema bancário em Python, usando herança.

Primeiro, criamos a classe **Conta** que vai ser a superclasse, o escopo, o esqueleto de toda conta desse banco. Ele tem um atributo, o *saldo*, que representa o valor na conta e o método **getSaldo** que retorna esse valor. Ela inicia o saldo com valor nulo.

Depois, criamos a classe **PF** para representar as contas de pessoas físicas. Ela é subclasse da *Conta* e tem um método diferente, o **setSaldo**, que faz a movimentação bancária e cobra cinco reais de cada operação.

Depois, criamos a classe **PJ**, para pessoas jurídicas, ela é igual a *PF*, porém seu método **setSaldo** cobra dez reais de cada operação. Chamamos a

superclasse no método `__init__` de ambos tipos de conta, para inicializar o atributo *saldo*, que é sempre inicialmente 0.

Então, fazemos nosso script rodar. Inicialmente ele vai te perguntar se quer abrir uma conta PF ou PJ, dependendo do que o usuário escolher, instanciamos um objeto da classe PF ou PJ.

Depois, em um looping infinito (um caixa eletrônico que funciona pra sempre), pergunta se você deseja ver o saldo ou fazer uma movimentação.

Para ver, basta chamar o método **getSaldo** do objeto *conta*.

Para sacar, basta fornecer um valor negativo e para depositar, um valor positivo.

Note que se for PF, vai ter uma taxa de 5 reais para cada movimentação. Se for PJ, essa taxa é de 10 reais.

Veja nosso script:

```
class Conta:
    def __init__(self):
        self.saldo = 0

    def getSaldo(self):
        return self.saldo

class PF (Conta):
    def __init__(self):
        Conta.__init__(self)

    def setSaldo(self, valor):
        self.saldo += valor - 5

class PJ (Conta):
    def __init__(self, inicial):
        Conta.__init__(self, inicial)

    def setSaldo(self, valor):
        self.saldo += valor - 10

print("1.Criar conta Pessoa Física")
print("2.Criar conta Pessoa Jurídica")
op = int(input("Opção:"))
```

```
if op==1:
    conta = PF()
elif op==2:
    conta = PJ()

while True:
    print("1. Ver saldo")
    print("2. Sacar / Depositar")
    op = int(input("Opção:"))

    if op == 1:
        print("R$", conta.getSaldo())
    elif op==2:
        val = float(input("Valor para movimentar:"))
        conta.setSaldo(val)
```

Rode ele e brinque um pouco, testando ambos tipos de contas.

Polimorfismo

- **Polimorfismo - O que é?**

Polimorfismo, em Python, é a capacidade que uma subclasse tem de ter métodos com o mesmo nome de sua superclasse, e o programa saber qual método deve ser invocado, especificamente (da super ou sub).

Ou seja, o objeto tem a capacidade de assumir diferentes formas (polimorfismo).

Vamos criar a classe **Superclasse** que tem apenas um método, o *hello()*. Instanciamos um objeto e chamamos esse método:

```
class Super:
    def hello(self):
        print("Olá, sou a superclasse!")

teste = Super()
teste.hello()
```

Agora o resultado:

- Olá, sou a superclasse!

Agora vamos criar outra classe, a **Sub**, que vai herdar a *Superclasse* e vamos definir nela um método de mesmo nome *hello()*, mas com um texto diferente:

```
class Super:
    def hello(self):
        print("Olá, sou a superclasse!")

class Sub(Super):
    def hello(self):
        print("Olá, sou a subclasse!")

teste = Sub()
teste.hello()
```

O resultado vai ser:

- Olá, sou a subclasse!

Veja bem, *Sub* herda a *Superclasse*, ou seja, tudo que nem na superclasse (atributos e métodos), vai ter na subclasse.

Porém, quando chamamos o método **hello()**, ele vai invocar o método da *subclasse* e não da superclasse! O Python entende: "Opa, ele instanciou um objeto da *subclasse*. Por isso vou invocar o método da *subclasse* e não da *superclasse*"

Ou seja, seu objeto assumiu a forma da *subclasse*, embora ele também seja uma *superclasse*.

Dizemos que o método da subclasse fez uma **sobreposição**, ele sobrepôs, passou por cima, do método da superclasse.

Vamos mais além agora, e criar a **Subsubclasse**, que vai herdar a **Sub**. Hora, se a *Subsubclasse* herda a *Sub*, e a *Sub* herda a *Super*, então a *Subsubclasse* também herda tudo da *Super*.

Porém, quando instanciamos um objeto da *Subsub* e invocamos o método *hello()*, ele vai rodar o método da *Subsub*:

```
class Super:
    def hello(self):
        print("Olá, sou a superclasse!")

class Sub (Super):
    def hello(self):
        print("Olá, sou a subclasse!")

class Subsub (Sub):
    def hello(self):
        print("Olá, sou a subsubclasse!")

teste = Subsub()
teste.hello()
```

Sacaram a lógica da coisa?

Embora estejamos chamando sempre o mesmo método (*hello*), que está presente em todas as classes e nas herdadas, ele age de maneira diferente!

GUI – Interface Gráfica

Até o momento, durante todo nosso curso, criamos *scripts* que rodavam no terminal de comando do Python. Porém, no dia-a-dia, não utilizamos muito terminal de comando.

Você liga seu computador, e aparece um campo para digitar seu login e senha. Aparece botões, você clica em coisas pra minimizar, maximizar, clica em uma aba, outra, abre um programa, fecha outro, clica aqui, clica acolá...

Veja este navegador que está lendo este tutorial ou lendo o PDF, tem botões lá em cima, menus, campos, listas de opções, barra de rolagem etc etc.

Ou seja, interagimos com *interfaces gráficas*, através de mouse e de teclado. E é justamente isto que iremos aprender nesta seção de nosso **Curso de Python**, a programar usando as *GUI - Graphic User Interfaces*.

O Módulo Tkinter

- **Tkinter - GUI em Python**

O Python por si só não possui um sistema próprio, nativo, para programação GUI.

Porém, existe um módulo muito famoso, bem poderoso e fácil de usar, chamado **Tkinter**.

Tkinter vem de *Tk interface*.

Tk é de *Tool Kit*, ou seja, um conjunto de ferramentas, usando a *TCL (Tool Command Language)*.

O módulo Tkinter é tão famoso, que é praticamente um padrão da linguagem Python, para programarmos interface gráfica. Porém, esse módulo é tão bacana que é usado por diversas outras linguagens de programação.

Componentes da Tk

GUI é composto por diversos componentes, chamados de *widget* (*windows gadget*, ou seja, componentes de janela), que nada mais são que objetos que o usuário vai ver e interagir.

O módulo *Tkinter* possui 15 *widgets*:

- **Button**: botões, onde algum evento ocorre quando clicamos
- **Checkbutton** : aqueles botões de *check*, tipo *on* e *off*
- **Radiobutton**: botões do tipo *radio*, onde é permitido escolher apenas uma opção (Masculino ou Feminino, por exemplo)
- **Canvas**: uma área retangular, que podemos colocar e exibir coisas dentro, como uma imagem
- **Entry**: um campo de entrada, onde o usuário pode digitar alguma linha de informação
- **Frame**: um 'container', uma área que serve pra abrigar e agrupar outros *widgets*
- **Label**: rótulo, uma área que exibe um texto ou uma imagem
- **Listbox**: uma lista de opções para o usuário marcar o que quiser
- **Menu**: uma lista de opções, um menu, onde o usuário vai clicar em alguma opção e algo vai ocorrer
- **Menubutton**: menu que é exibido na tela e pode ser clicado pelo usuário

- **Message**: mostra múltiplas linha de texto na tela
- **Scale**: um widget onde o usuário pode clicar, segurar e mover um ao longo de uma faixa (tipo, aumentar e reduzir o volume)
- **Scrollbar**: barra de rolagem
- **Text**: permite o usuário digitar múltiplas linhas de texto
- **Toplevel**: um container, como um *Frame*, mas exibe sua própria janela

• Como usar o Tkinter - Criando uma GUI Simples

Vamos criar o exemplo mais simples possível de programa usando interface gráfica, que é simplesmente criar uma janela, vazia, sem nada:

```
import tkinter
```

```
main_window = tkinter.Tk()
tkinter.mainloop()
```

O comando acima funciona para Python 3.

Se estiver rodando o Python 2, use com T maiúsculo:

```
import Tkinter
```

```
main_window = Tkinter.Tk()
Tkinter.mainloop()
```

O resultado é:



Primeiro, importamos o módulo *tkinter*.

Depois, criamos uma classe do tipo **Tk()** e atribuímos a variável *main_window* (janela principal).

Por fim, chamamos a função **mainloop()**, do módulo. Ela é responsável por ficar exibindo, indefinidamente essa janelinha principal, até que você feche ela.

- **Tkinter e Orientação a Objetos**

O normal, o comum e recomendável, é trabalhar com programação GUI usando os conceitos que aprendemos [programação orientada a objetos](#), pois assim fica bem mais fácil entender.

Vamos ver esses componentes como *objetos*.
Veja como fica em POO:

```
import tkinter
```

```
class MinhaGUI:
```

```
    def __init__(self):  
        # Criando a janela principal  
        main_window = tkinter.Tk()  
  
        # Exibindo ela em looping  
        tkinter.mainloop()
```

```
minha_gui = MinhaGUI()
```

Não é a toa que a seção de orientação a objetos veio antes da seção de GUI.

Label em GUI - Exibindo um texto

- **Como Exibir um Texto - Label**

Agora que já estudamos um pouco de GUI e o módulo Tkinter do Python, vamos aprender a usar o *widget* mais simples e fácil de todos o *Label*, que serve para exibir um simples texto.

Para exibir um simples texto, vamos usar a função **Label()** do módulo *tkinter*. O primeiro argumento tem que ser a janela (*main_window*, do tipo *Tk()*) onde o *label* vai ficar e o segundo argumento é a variável **text** com a string que queremos exibir.

Vamos salvar esse *Label* na variável **texto**.

Depois, invocamos o método **pack()** da **Label()**, para posicionar e exibir o *label* na janela.

Por fim, chamamos a *mainloop()* para exibir nossa janela.

Nosso código fica:

```
import tkinter
```

```
class MinhaGUI:
```

```
    def __init__(self):
```

```
        # Criando a janela principal
```

```
        self.main_window = tkinter.Tk()
```

```
        # Criando um Label com o texto 'Curso Python Progressivo'
```

```
        self.texto = tkinter.Label(self.main_window, text='Curso Python  
Progressivo! ' )
```

```
        # Chamando o metodo pack() da função Label()
```

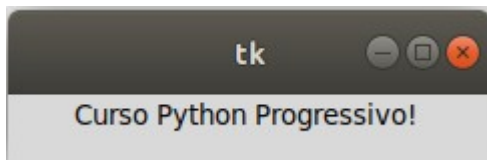
```
        self.texto.pack()
```

```
        # Fazer o Tkinter exibir o looping da janela
```

```
        tkinter.mainloop()
```

```
minha_gui = MinhaGUI()
```

O resultado é a janela com o rótulo (*label*):



- O método ***pack()*** e o argumento ***side***

Se você maximizar a janela que vai abrir, vai perceber que a linha de texto fica bem lá em cima, e no centro. Esse é o padrão, mas podemos mudar isso através do método ***pack()*** que recebe o argumento *side* com as seguintes opções:

1. *side*='left'
2. *side*='right'
3. *side*='bottom'
4. *side*='top'

Por exemplo, vamos criar um novo *label* na variável ***label2*** e chamar de ***label1*** a primeira, que exibimos a string 'Curso Python Progressivo!'

Vamos fazer com que *label1* fique no topo e *label2* fique sempre lá embaixo, basta usar o argumento *side* no método ***pack()***, veja como fica o código:

```
import tkinter
class MinhaGUI:
    def __init__(self):
        # Criando a janela principal
        self.main_window = tkinter.Tk()

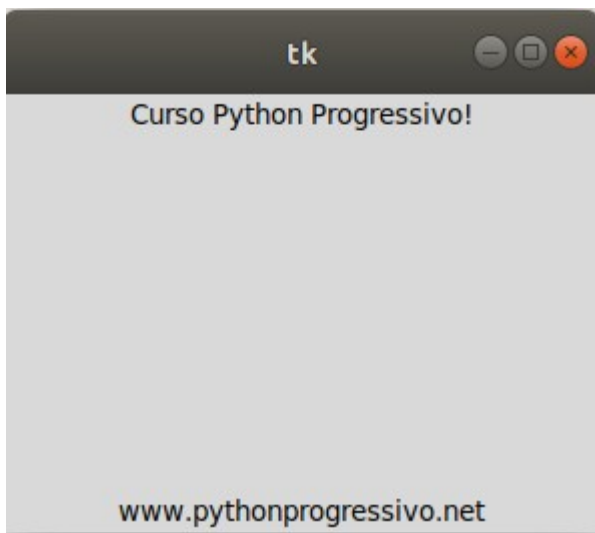
        # Criando os labels
        self.label1 = tkinter.Label(self.main_window, text='Curso Python
Progressivo!')
        self.label2 = tkinter.Label(self.main_window,
text='www.pythonprogressivo.net' )

        # Exibindo os labels
        self.label1.pack(side='top')
        self.label2.pack(side='bottom')

        # Fazer o Tkinter exibir o looping da janela
        tkinter.mainloop()

minha_gui = MinhaGUI()
```

E o resultado:



Note que você pode maximizar, diminuir ou aumentar a janela, mas o *label1* fica sempre lá em cima e o *label2* fica sempre lá embaixo, quem 'cuida' disso é o método **pack()**.

Bacana, não é?

- A Função **Label()** e seus Parâmetros

Existem outros parâmetros para passarmos para a função **Label** além da janela principal o *text* com a string a ser exibida.

Por exemplo, o parâmetro **bg** recebe a cor de background (plano de fundo). O parâmetro **fg**, recebe a cor da fonte.

O seguinte código:

```
import tkinter
```

```
class MinhaGUI:
```

```
    def __init__(self):
```

```
        # Criando a janela principal
```

```
        self.main_window = tkinter.Tk()
```

```
        # Criando a label Ceará e exibindo
```

```
        self.label = tkinter.Label(self.main_window, text="Ceará",
```

```
bg="black", fg="white")
```

```

self.label.pack(fill = tkinter.X)

# Criando a label Flamengo e exibindo
self.label = tkinter.Label(self.main_window, text="Flamengo",
bg="red", fg="black")
self.label.pack(fill = tkinter.X)

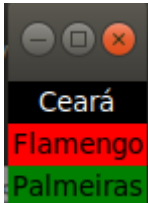
# Criando a label Palmeiras e exibindo
self.label = tkinter.Label(self.main_window, text="Palmeiras",
bg="green", fg="black")
self.label.pack(fill = tkinter.X)

# Fazer o Tkinter exibir o looping da janela
tkinter.mainloop()

minha_gui = MinhaGUI()

```

Tem o seguinte resultado:



O parâmetro *fill* é de preenchimento, usamos **tkinter.X** para expandir (*expand*) e ocupar toda a tela.
Entenderam a lógica da coisa?

Cria uma janela (*main_window* ou qualquer nome que queira), é uma instância de **Tk()**

Cria os *label* com a função **Label** e depois exibe com a **pack()**.

Por fim, chama a **mainloop()** pra a exibição continuar pra sempre, até você clicar em fechar.

Frame - Organizando os widgets

Neste tutorial de Python, vamos aprender o que são os Frames, para que servem e como usar, em programação gráfica (GUI).

- **Frames em Python**

Um *frame*, em programação GUI, nada mais é que uma espécie de 'container', cujo propósito principal é armazenar e agrupar outros *widgets*.

Na barra do seu navegador, por exemplo, lá em cima, tem um *frame* com as várias opções de menu (Editar, Exibir, Ajuda etc).

Logo abaixo, deve ter outro frame, que agrupa vários botões: Página inicial, Parar, Atualizar, Voltar, Ir pra frente etc.

Ou seja, frame é uma 'região' em sua aplicação gráfica, que você vai colocar uma porção de outras coisas. É um widget que serve pra armazenar outros widgets.

O normal é colocar vários widgets num frame, e organizar eles ali dentro (cores, disposição, eventos etc). Depois, cria outro frame pra exibir na mesma janela, mas com outros widgets, com sua própria maneira de arranjo.

Abaixo, dois frames, lado a lado:



Note que, internamente, cada um tem seu arranjo (botões, disposições, cores, alinhamentos etc)

- **Como usar Frames em Python**

Vamos fazer uma aplicação que vai exibir dois frames.

Em cada um dos frames, vais colocar um [Label](#), com um texto simples em cada um.

Primeiro, damos um **import** em tudo (*) do módulo *tkinter*.

Nossa classe que vai criar o programinha, se chama *MinhaGUI*.

Primeiro, instanciamos o objeto da Tk(), que é nossa janela principal.

Depois, vamos criar dois frames: o *frame_cima* e o *frame_baixo*, através da classe **Frame()**, que tem que recebe como argumento um objeto do tipo Tk(), no nosso caso é o objeto *janela_principal*, que é onde os frames estarão localizados (janela mãe).

A seguir, criamos dois labels, o *label1* e o *label2*, e passamos os frames criados como argumento, um label para cada frame.

Damos o *pack()* pra posicionar os labels nos respectivos frames.

E por fim, damos o *pack()* para posicionar os frames (sim, frames também precisam ser posicionados nas janelas).

Depois é só chamar a função **mainloop()** que vai fazer sua aplicação rodar bonitinha.

Código Python:

```
from tkinter import *
```

```
class MinhaGUI:
```

```
    def __init__(self):
```

```
        # Criando a janela principal
```

```
        self.janela_principal = Tk()
```

```
        # Criando os frames
```

```
        self.frame_cima = Frame(self.janela_principal)
```

```
        self.frame_baixo = Frame(self.janela_principal)
```

```
        # Criando os labels
```

```
        self.label1 = Label(self.frame_cima, text='To no frame de cima!')
```

```
        self.label2 = Label(self.frame_baixo, text='To no frame de baixo!')
```

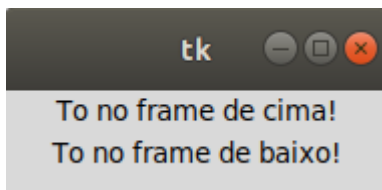
```
# Posicionando os labels nos frames
self.label1.pack(side='top')
self.label2.pack(side='top')

# Posicionando o frame
self.frame_cima.pack()
self.frame_baixo.pack()

# Fazer o Tkinter exibir o looping da janela
mainloop()
```

```
minha_gui = MinhaGUI()
```

Resultado:



- **Parâmetros da Frame**

No exemplo anterior, passamos apenas uma informação ao criar um objeto do tipo Frame, que foi sua janela mestre. É o único parâmetro obrigatório, é o *master*.

Existem alguns outros, como o **bg**, (pode usar **background** também, o tkinter entende ambos) que define a cor de fundo do frame.

O parâmetro **height** define a altura e o **width** a largura, em pixels, de cada frame.

Por exemplo, o código Python a seguir:

```
from tkinter import *

class MinhaGUI:
    def __init__(self):
        # Criando a janela principal
        self.janela_principal = Tk()

        # Criando os frames
```



```

        self.frame_cima = Frame(self.janela_principal, bg="white",
height=70, width=400)
        self.frame_baixo = Frame(self.janela_principal, bg='red', height=70,
width=400)

        # Posicionando o frame
        self.frame_cima.pack()
        self.frame_baixo.pack()

        # Fazer o Tkinter exibir o looping da janela
        mainloop()

minha_gui = MinhaGUI()

```

Cria a seguinte aplicação:



Se passar o argumento **side="left"** nas chamadas da *pack()*, essas duas faixas vão aparecer lado a lado ao invés de uma em cima da outra.

• Exercício de Python

Busque na internet a figura da bandeira da França e da Holanda.

Código Python:

```

from tkinter import *

class MinhaGUI:
    def __init__(self):
        # Criando as janelas
        self.janela1 = Tk()
        self.janela2 = Tk()

```

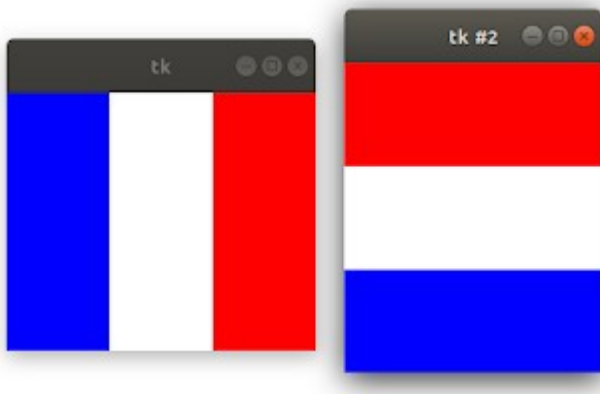
```
# Bandeira da França
self.frame1 = Frame(self.janela1, bg="blue", height=200, width=80)
self.frame2 = Frame(self.janela1, bg='white', height=200, width=80)
self.frame3 = Frame(self.janela1, bg='red', height=200, width=80)
self.frame1.pack(side="left")
self.frame2.pack(side="left")
self.frame3.pack(side="left")

# Bandeira da Holanda
self.fram1 = Frame(self.janela2, bg="red", height=80, width=200)
self.fram2 = Frame(self.janela2, bg='white', height=80, width=200)
self.fram3 = Frame(self.janela2, bg='blue', height=80, width=200)
self.fram1.pack(side="top")
self.fram2.pack(side="top")
self.fram3.pack(side="top")

# Fazer o Tkinter exibir o looping da janela
mainloop()
```

```
minha_gui = MinhaGUI()
```

Resultado:



Note que dessa vez criamos duas janelas, então quando rodamos o script, aparecem duas janelas!

Caixa de diálogo: messagebox

- **Caixa de Mensagem: *info dialog box***

Caixa de diálogo ou simplesmente caixa de mensagem, nada mais é que uma simples janelinha que aparece, com algum texto e um botão de OK, que fecha a *box* quando clicamos.

Para usarmos, primeiro importamos o **messagebox** do módulo **tkinter**. Depois, usamos a função **showinfo**, que recebe duas strings como argumento: a primeira vai no título da caixa de diálogo que vai abrir, a segunda string é a mensagem que vai aparecer.

Veja um exemplo de código:

```
from tkinter import messagebox
```

```
messagebox.showinfo('Python Progressivo', \n                    'Adoro o curso Python Progressivo')
```

O resultado é:



Agora, ao invés de usar a função *showinfo*, tente as seguintes:

- `showwarning()`
- `showerror()`
- `askquestion()`
- `askokcancel()`
- `askyesno ()`
- `askretrycancel ()`

Tente todas mesmo! O que aconteceu? Bacana e simples, não é?

Note que ao clicarmos nos botão de Ok, Cancelar, Tente novamente etc, nada acontece. Mas, calma, no próximo tutorial vamos aprender como fazer as coisas acontecerem ao clicarmos em algum botão.

Botão em GUI

Neste tutorial de GUI em Python, vamos aprender a criar e usar um dos widgets mais importantes, o *button*, ou seja, vamos criar botões em nossos programas.

- **Widget de Botão: `Button`**

O widget *Button* nada mais é que aquele velho e conhecido botão, nas janelas, que clicamos em cima e acontece alguma coisa.

Em programação gráfica em Python, o que ocorre por trás dos panos quando criamos um botão é a chamada de um método.

Ou seja, clicou em **OK**, então o código vai atrás de um determinado método ou função e executa ele. Já se clicar em **Cancelar** outro método ou função que será invocado.

Para criarmos um botão, usamos a classe *Button*, que recebe três parâmetros:

1. A janela mestre (objeto do tipo `Tk()`)
2. `text`: string com o texto que vai aparecer no botão (Ok, Cancelar, Sair...)
3. `command`: nome do método ou função que vai ser chamado quando clicarmos no botão, mas sem os parêntesis

- **Exemplo de uso do Botão em Python**

Vamos criar um botão que com o texto "Clique aqui", que quando cli.cado, chama o método *hello_world()*, que dentro dele exibe uma [caixa de diálogo](#).

Veja como fica nosso código Python:

```
from tkinter import *
from tkinter import messagebox

class MinhaGUI:
    def __init__(self):
        # Criamos a janela principal
        self.janela_principal = Tk()

        # Criando o botão
```

```

        self.botao = Button(self.janela_principal, text='Clique aqui',
command=self.hello_world)

        # Empacotando o botão na janela principal
        self.botao.pack()

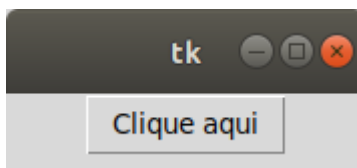
        # Rodando
        mainloop()

def hello_world(self):
    messagebox.showinfo('Adoro a Apostila Python Progressivo!')

```

gui = MinhaGUI()

E o resultado:



Ao clicar, aparece essa caixa de diálogo:



O Botão sair: *quit button*

Se você clicar em "Clique aqui", aparece a caixa de diálogo.

Se clicar no "Ok" da caixa de mensagem, volta pra janela principal, que tem o "Clique aqui" e fica nisso, eternamente. Só acaba se clicar no x de fechar.

Mas o comum é ter um botão de fechar, cancelar, sair...que ao clicar nele, nossa aplicação se encerre.

Pois bem, todo objeto da classe **Tk()** tem um método chamado *quit()*, mas chamar ele que as janelas fechem ao clicar no botão.

Então, vamos criar mais um botão, o *botao_sair*, que ao ser clicado (*command*) chama o método **quit()** da janela principal, que fecha o programa.

Veja como fica nosso código:

```
from tkinter import *
from tkinter import messagebox

class MinhaGUI:
    def __init__(self):
        # Criamos a janela principal
        self.janela_principal = Tk()

        # Criando os botões
        self.botao = Button(self.janela_principal, text='Clique aqui',
command=self.hello_world)
        self.botao_sair = Button(self.janela_principal, text='Sair',
command=self.janela_principal.quit)

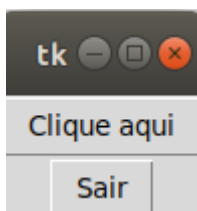
        # Empacotando os botões janela principal
        self.botao.pack()
        self.botao_sair.pack()

        # Rodando
        mainloop()

    def hello_world(self):
        messagebox.showinfo('Adoro a Apostila Python Progressivo!')
```

```
gui = MinhaGUI()
```

E o resultado:



Experimente clicar em Sair e veja o que acontece!

Entry widget – Recebendo dados do usuário

Neste tutorial de programação gráfica GUI em Python, vamos aprender o que é e como usar o widget *Entry*.

- **Recebendo informações do usuário**

Até o momento, nossa única interação código-usuário foi através do clique do usuário em botões, seja em nossos [buttons](#) como em [caixas de diálogo](#).

Porém, a forma mais comum de interação entre um programa e o usuário, é através daquelas áreas retangulares, onde escrevemos algum texto.

Por exemplo, para preencher o campo de nome, telefone, CEP, data de nascimento etc.

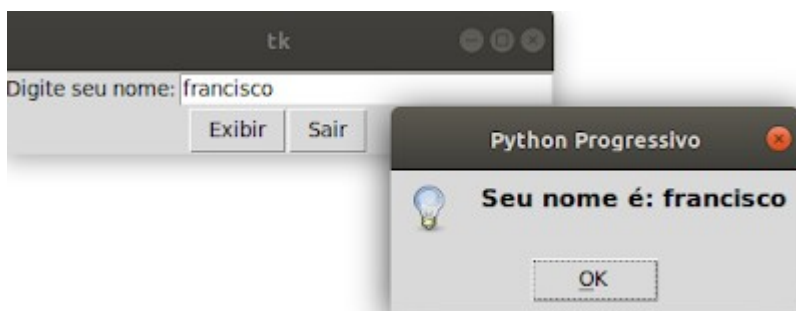
Geralmente, o normal é termos algo do tipo:

Um label com algum texto

Um campo de entrada, para escrevermos algo

Um botão, para fazer algo com o dado que escrevemos

Vejamos um print de uma aplicação real:



- **Como usar entradas: *Entry* widget**

Vamos aprender como fazer o programinha do print anterior.

Na primeira janela, temos dois frames:

frame de cima: tem um *label* e um widget do tipo *Entry*

frame de baixo: dois botões

Após criar os objetos do tipo label e os frames, vamos criar nosso widget *Entry*.

Ele recebe dois parâmetros:

1. O container que ele vai estar (frame ou janela)
2. width: número de caracteres que vai ser possível digitar no campo de texto

Vamos usar **width=30**

Em seguida, empacotamos o label e *entry* no frame de cima, lado a lado (side='left').

Nosso *entry* vai se chamar **entrada** e ele tem um método especial, o **get()**, que retorna uma string com o que foi digitado na caixa.

Em seguida, criamos dois botões.

O primeiro chama o método **exibe()**, onde esse método exibe uma string. Por fim, um botão de *quit*.

Agora vamos empacotar esse botões no frame de baixo.

Depois empacotamos os dois frames na janela principal...e rodamos o programa!

Veja como fica o código completo:

```
from tkinter import *
from tkinter import messagebox

class MinhaGUI:
    def __init__(self):
        # Criamos a janela principal
        self.janela_principal = Tk()

        # Criando os frames
        self.frame_cima = Frame(self.janela_principal)
        self.frame_baixo = Frame(self.janela_principal)

        # Criando label e botões do frame de cima
        self.label = Label(self.frame_cima, text='Digite seu nome:')

        # Criando o widget de entrada
        self.entrada = Entry(self.frame_cima, width=30)
```



```

# Empacotando label e entrada no frame de cima
self.label.pack(side='left')
self.entrada.pack(side='left')

# Criando os botões, no frame de baixo
self.botoe = Button(self.frame_baixo, text='Exibir',
command=self.exibe)
self.botoe_sair = Button(self.frame_baixo, text='Sair',
command=self.janela_principal.quit)

# Empacotando os botões no frame de baixo
self.botoe.pack(side='left')
self.botoe_sair.pack(side='left')

# Empacotando os botões janela principal
self.botoe.pack()
self.botoe_sair.pack()

# Empacotando os frames na janela principal
self.frame_cima.pack()
self.frame_baixo.pack()

# Rodando
mainloop()

def exibe(self):
    messagebox.showinfo('Python Progressivo', 'Seu nome é: '+
self.entrada.get())

gui = MinhaGUI()

```

Ou seja, o pulo do gato dos widgets *Entry* é o método **get()** que todo objeto desse tipo tem.

Esse método retorna SEMPRE uma string, com o que foi digitado.

Quer pegar um número? Ok, mas primeiro converte de string pra inteiro ou float, só depois usa o número. Mas é sempre uma string, ok?

StringVar – Label como saídas dinâmicas

Neste tutorial de nosso **Curso de Python**, vamos aprender a usar a classe *StringVar*, que nos permite criar labels dinâmicas, que usaremos como saída de dados.

- **Label como saída de dados dinâmica**

Até o momento, em nossos estudos em programação gráfica, viemos usando as caixas de diálogo para exibir alguma informação, uma saída de dados, como uma simples string.

Porém, não é isso que ocorre na prática, não fica sempre pipocando janelinhas, até porque isso é chato. É normal que, você insira um dado, aperte um botão ou não, e automaticamente já apareça uma informação na janela mesmo, sem nenhuma nova janela surgindo.

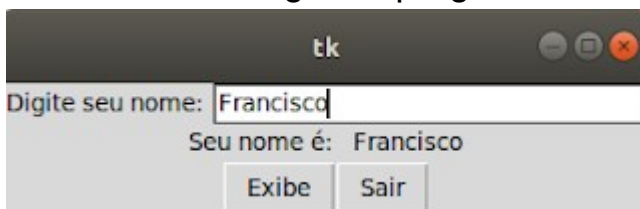
É como se fosse uma espécie de *label* dinâmica. Inicialmente ela mostra: *Seu IMC é:*

Então você informa seu peso e altura, e esse label se transforma;
Seu IMC é: 26, você está com sobrepeso

Essa mudança deve ocorrer simplesmente clicando em algum botão. Vamos aprender como fazer isso?

- **Como usar StringVar**

Vamos criar o seguinte programa:



Digitamos nosso nome, clicamos em Exibe e aparece o label embaixo: "Seu nome é: ..."

Esse programa é composto por três frames:

Frame de cima: tem o label "Digite seu nome" e o campo de entrada

Frame do meio: tem o label "Seu nome é:" e o label dinâmico (*StringVar*), que exibe o nome que o usuário digitou no campo de entrada
Frame de baixo: dois botões, o Exibe e o Sair

O primeiro e o último frame, bem como seus widgets internos, sabemos muito bem como criar e trabalhar com eles.

Vamos focar no frame do meio, que tem a label dinâmica.

O segredo aqui é criar um objeto do tipo *StringVar()*, vamos chamar de **label_dinamica**.

Ela que vai ficar 'pegando' o que é digitado no campo de entrada. Mas ela pega e passa pra outra label, a **label3**.

A **label3** recebe um parâmetro chamado *textvariable*, que como o nome diz, vai receber um texto variável. Como argumento, passamos o nome do label dinâmica, que no nosso caso é **label_dinamica**.

O código fica assim:

- `self.label3 = Label(self.frame_meio, textvariable=self.label_dinamica)`

Ou seja, essa linha de código quer dizer: crie uma label de nome *label3*, que vai ficar no *frame_meio* e o valor que ela vai exibir é um texto variável, que a *label_dinamica* vai ficar pegando).

Essa label dinâmica vai pegar o valor escrito na entrada, quando clicamos no botão **Exibe**.

Quando isso ocorre, esse botão chama o método **exibe()**

Dentro desse método, pegamos o valor que foi digitado e armazenamos na variável **nome** da seguinte maneira:

```
nome = self.entrada.get()
```

Agora, vamos passar essa variável para a *label dinâmica*, através do método **set()** que ela tem, que recebe uma string:

- `self.label_dinamica.set(nome)`

Veja como fica nosso código Python:

```
from tkinter import *
```

```
class MinhaGUI:
```

```
    def __init__(self):
```

```
        # Cria janela principal
```

```
        self.janela_principal = Tk()
```

```
        # Cria três frames
```

```
        self.frame_cima = Frame(self.janela_principal)
```

```
        self.frame_meio = Frame(self.janela_principal)
```

```
        self.frame_baixo = Frame(self.janela_principal)
```

```
        # No frame de cima, um label e uma entry
```

```
        self.label1 = Label(self.frame_cima, text='Digite seu nome: ')
```

```
        self.entrada = Entry(self.frame_cima, width = 30)
```

```
        # Empacotando o label e entrada no primeiro frame
```

```
        self.label1.pack(side='left')
```

```
        self.entrada.pack(side='left')
```

```
        # Labels do frame do meio
```

```
        self.label2 = Label(self.frame_meio, text='Seu nome é: ')
```

```
        self.label_dinamica = StringVar()
```

```
        self.label3 = Label(self.frame_meio,
```

```
        textvariable=self.label_dinamica)
```

```
        # Empacotando as labels do meio
```

```
        self.label2.pack(side='left')
```

```
        self.label3.pack(side='left')
```

```
        # Criando os botões
```

```
        self.botao = Button(self.frame_baixo, text='Exibe',  
command=self.exibe)
```

```
        self.botao_sair = Button(self.frame_baixo, text='Sair',  
command=self.janela_principal.quit)
```

```
        # Empacotando botões e label
```

```
        self.botao.pack(side='left')
```

```
        self.botao_sair.pack(side='left')
```

```
        # Empacotando os frames na janela principal
```

```
        self.frame_cima.pack()
```

```
self.frame_meio.pack()  
self.frame_baixo.pack()
```

```
# Rodando  
mainloop()
```

```
def exibe(self):  
    nome = self.entrada.get()  
    self.label_dinamica.set(nome)
```

```
gui = MinhaGUI()
```

Rode ele! Leia linha por linha e tente entender tudo que está se passando aí.

Botão de Rádio – Radiobutton e IntVar

Neste tutorial de [GUI em Python](#), vamos aprender como criar os *radio buttons*, aqueles botões de rádio, onde escolhemos apenas uma opção dentre várias:

☒ Radio1

☐ Radio2

☐ Radio3

- **Botões de Rádio: *Radio Button***

Botões de rádio são aquelas opções, que aparecem em grupo, para você poder marcar apenas uma alternativa. Quando você marca uma opção, automaticamente as outras estarão desmarcadas.

Ou seja, dizemos que um botão de rádio é mutuamente exclusivo: só pode marcar uma opção e apenas uma, nada mais nem nada menos que uma.

Eles são bem úteis quando o programa deseja saber o que o usuário quer fazer, deixando ele escolher apenas uma opção.

- **RadioButton - Como usar em Python**

Para criarmos um grupo de *radio buttons* em Python, devemos fazer duas coisas:

Criar os botões, onde cada um é objeto da classe *Radiobutton*

Um objeto da classe *IntVar*, que vai armazenar o valor do botão selecionado

Para criar um objeto da classe *Radiobutton*, passamos como argumento:

O container onde o botão de rádio vai (geralmente um frame)

1. **text** - variável que armazena a string que vai aparecer ao lado do botão de rádio
2. **variable** - nome da variável, que é um objeto da classe tipo **IntVar**

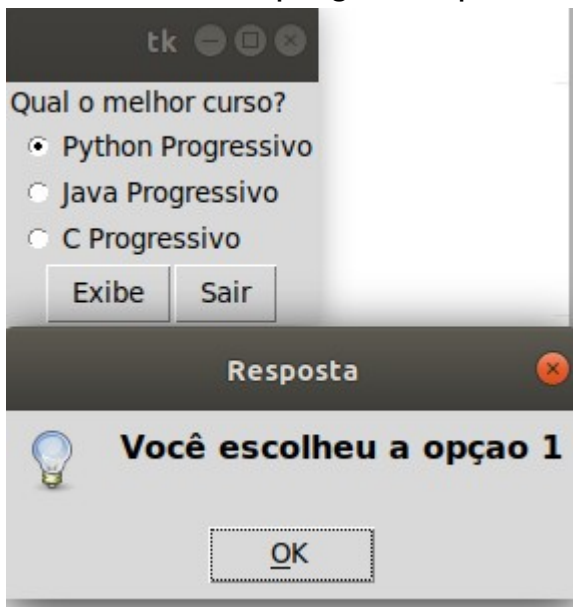
3. **value** - valor inteiro, onde cada opção deve corresponder a um número diferente

Pegamos o valor selecionado através do método **get()** de objetos da classe *IntVar*.

Vamos ver um exemplo real, funcionando e acontecendo, para entender melhor ?

- **Exemplo de uso de um Radiobutton**

Vamos criar um programa que vai ter a seguinte 'cara':



Primeiro, criamos nossa *janela_principal*, depois dois frames: *frame_cima* e o *frame_baixo*.

No frame de cima, vamos colocar o label e os botões de rádio.

No frame de baixo, vão os botões **Exibe** e **Sair**.

Depois, criamos o objeto **radio_valor**, da classe *IntVar*, que vai armazenar o valor do botão de rádio selecionado. Inicialmente setamos ele como 1, para aparecer marcado a opção de valor 1.

Em seguida criamos o label "Qual melhor curso?".

Depois, os três *Radiobutton*. Vamos colocar todos no *frame_cima*, todos vão usar a variável **radio_valor** e cada um tem um número inteiro diferente, de 1 até 3, para sabermos qual valor foi marcado.

Empacotamos o label e os botões no frame de cima.

Depois criamos os botões **Exibe** e o **Sair**. O **exibe** pega o valor assinalado (que está no método `get()` do objeto `radio_valor` da classe `VarInt`) e **exibe** a opção que escolhemos.

Veja como ficou nosso código Python:

```
from tkinter import *
from tkinter import messagebox

class MinhaGUI:
    def __init__(self):
        # Cria janela principal
        self.janela_principal = Tk()

        # Cria dois frames
        self.frame_cima = Frame(self.janela_principal)
        self.frame_baixo = Frame(self.janela_principal)

        # Objeto IntVar dos botões
        self.radio_valor = IntVar()
        self.radio_valor.set(1) # Para a primeira opção ficar marcada

        # Criando os radio buttons e o label
        self.label = Label(self.frame_cima, text='Qual o melhor curso?')
        self.python = Radiobutton(self.frame_cima, text='Python
Progressivo', \
                                variable = self.radio_valor, value=1)
        self.java = Radiobutton(self.frame_cima, text='Java Progressivo', \
                                variable = self.radio_valor, value=2)
        self.c = Radiobutton(self.frame_cima, text='C Progressivo', \
                              variable = self.radio_valor, value=3)

        # Empacotando o label e os radio buttons
        self.label.pack(anchor = 'w')
        self.python.pack(anchor = 'w')
        self.java.pack(anchor = 'w')
        self.c.pack(anchor = 'w')

        # Criando os botões
        self.botao = Button(self.frame_baixo, text='Exibe',
command=self.exibe)
        self.botao_sair = Button(self.frame_baixo, text='Sair',
command=self.janela_principal.quit)
```



```
# Empacotando botões e label
self.botao.pack(side='left')
self.botao_sair.pack(side='left')

# Empacotando os frames na janela principal
self.frame_cima.pack()
self.frame_baixo.pack()

# Rodando
mainloop()
```

```
def exibe(self):
    nome = str(self.radio_valor.get())
    messagebox.showinfo('Resposta', 'Você escolheu a opção ' +
nome)
```

```
gui = MinhaGUI()
```

•

Chamando Métodos sem Botão

Se você puxar pela memória, vai se lembrar que muitas vezes nem precisar clicar em um botão, para o programa tomar alguma ação, basta selecionar um dos botões de rádio e algo já acontece.

Para fazer isso, basta adicionar o parâmetro **command** ao criar um objeto da classe *Radiobutton* com o nome do método que você deseja que seja executado.

Veja como fica nosso programa sem o uso de botões:

```
from tkinter import *
from tkinter import messagebox

class MinhaGUI:
    def __init__(self):
        # Cria janela principal
        self.janela_principal = Tk()

        # Cria dois frames
        self.frame_cima = Frame(self.janela_principal)
        self.frame_baixo = Frame(self.janela_principal)
```

```

# Objeto IntVar dos botões
self.radio_valor = IntVar()
self.radio_valor.set(1)      # Para a primeira opção ficar marcada

# Criando os radio buttons e o label
self.label = Label(self.frame_cima, text='Qual o melhor curso?')
self.python = Radiobutton(self.frame_cima, text='Python
Progressivo', \
                           variable = self.radio_valor, value=1,
command=self.exibe)
self.java = Radiobutton(self.frame_cima, text='Java Progressivo', \
                           variable = self.radio_valor, value=2,
command=self.exibe)
self.c = Radiobutton(self.frame_cima, text='C Progressivo', \
                           variable = self.radio_valor, value=3,
command=self.exibe)

# Empacotando o label e os radio buttons
self.label.pack(anchor = 'w')
self.python.pack(anchor = 'w')
self.java.pack(anchor = 'w')
self.c.pack(anchor = 'w')

# Empacotando os frames na janela principal
self.frame_cima.pack()
self.frame_baixo.pack()

# Rodando
mainloop()

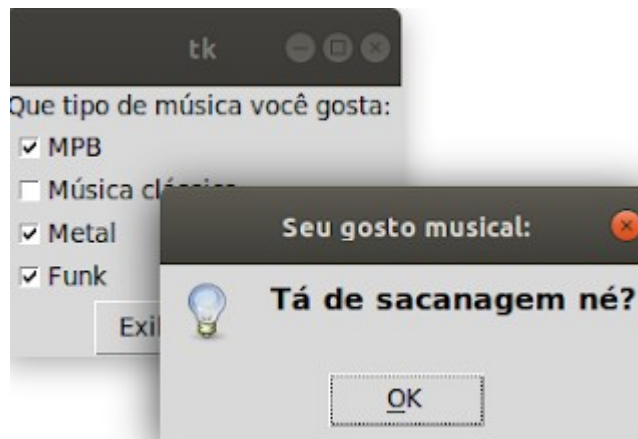
def exibe(self):
    nome = str(self.radio_valor.get())
    messagebox.showinfo('Resposta', 'Você escolheu a opção ' +
nome)

gui = MinhaGUI()

```

Botão de Checagem - A classe Checkbutton

Neste tutorial de GUI em Python, vamos estudar o widget de *Checkbutton*, os famosos botões de checagem, tipo *on/off*, de marcado ou não marcado:



- **Checkbutton em Python**

Se você entendeu bem o conceito e uso dos [botões de rádio \(radiobutton\)](#), vai facilmente entender os *checkbuttons*.

Assim como os botões de rádio, os de *check* podem ser selecionados e deselecionados (espero que essa palavra exista). A diferença é que num mesmo grupo de botões de rádio, você pode deixar apenas um selecionado.

Já nos botões de check, você pode selecionar quantos quiser, até mesmo todos ou nenhum.

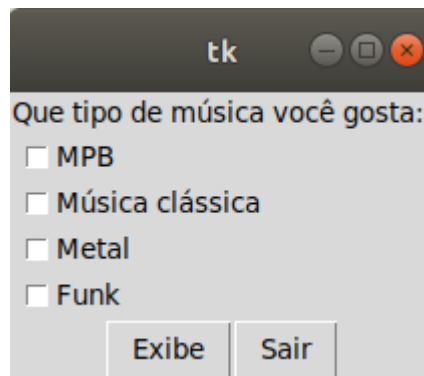
Para trabalharmos com botões de *check*, usamos a classe *Checkbutton*, do módulo *tkinter* também.

A grande diferença, na hora de programar, é que vamos usar um objeto da classe *IntVar* para cada botão de checagem. Nos botões de rádio, usávamos apenas um objeto *IntVar* para o grupo de botões, pois apenas um dele seria marcado...já nos de check, todos podem ser marcados, por isso cada botão tem seu *IntVar*.

- **Como usar Checkbutton em uma GUI em Python**

Mas vamos deixar de papo teórico e partir para a prática.

Vamos criar um programinha com a seguinte cara:



Teremos dois frames:

- frame_cima: o label 'Que tipo de música você gosta' e os 4 botões de check
- frame_baixo: os dois botões

Criamos 4 objetos do tipo *IntVar*: *checkvar1*, *checkvar2*, ..., *checkvar4*, um para cada *checkboxbutton*.

Depois colocamos o valor 0 como argumento do método *set* da classe *IntVar*, para dizer que queremos que os *check buttons* iniciem zerados (sem estarem marcados).

Depois, vamos criar nossos widgets. Primeiro o [label](#).

Depois, os 4 *checkboxbuttons*, através da classe *Checkbutton*, que recebe primeiro o container que vamos adicionar os botões (*frame_cima*), depois o texto que vai ao lado do *check* e o parâmetro **variable**, onde passamos o objeto do tipo *IntVar* responsável por lidar com o valor assinalado (0 para desmarcado ou 1 para marcado).

Empacotamos o label e os *checkboxbuttons* (usamos *anchor='w'* para alinhar a esquerda, *west*).

Depois criamos os [botões](#) **Exibe** e o **Sair**, onde o *Exibe* chama o método **exibe()**.

Empacotamos os botões, depois os frames.

Agora vamos para o método **exibe()**.

Ele vai exibir uma [caixa de diálogo](#) com uma string começada com 'Você curte: ' e vai adicionando 'MPB', 'Música clássica' e ou 'Metal' de acordo com o que for assinalado.

Exceto se você curtir Funk, se marcar essa opção vai aparecer a string 'Tá de sacanagem né?'

Para sabermos o valor que está nos objetos da *IntVar*, basta acessar o método **get()**.

Se estiver assinalado, ele retorna 1. Se não estiver, retorna 0.

Veja como ficou o código de nosso programinha:

```
from tkinter import *
from tkinter import messagebox

class MinhaGUI:
    def __init__(self):
        # Cria janela principal
        self.janela_principal = Tk()

        # Cria dois frames
        self.frame_cima = Frame(self.janela_principal)
        self.frame_baixo = Frame(self.janela_principal)

        # Objetos IntVar dos botões
        self.checkvar1 = IntVar()
        self.checkvar2 = IntVar()
        self.checkvar3 = IntVar()
        self.checkvar4 = IntVar()

        # Setando valor 0 para aparecem desmarcados
        self.checkvar1.set(0)
        self.checkvar2.set(0)
        self.checkvar3.set(0)
        self.checkvar4.set(0)

        # Criando os check buttons e o label
        self.label = Label(self.frame_cima, text='Que tipo de música você
gosta: ')
        self.checkbutton1 = Checkbutton(self.frame_cima, text='MPB', \
            variable = self.checkvar1)
        self.checkbutton2 = Checkbutton(self.frame_cima, text='Música
clássica', \
            variable = self.checkvar2)
        self.checkbutton3 = Checkbutton(self.frame_cima, text='Metal', \
            variable = self.checkvar3)
```

```

self.checkbutton4 = Checkbutton(self.frame_cima, text='Funk', \
                                variable = self.checkvar4)

# Empacotando o label e os check buttons
self.label.pack(anchor = 'w')
self.checkbutton1.pack(anchor = 'w')
self.checkbutton2.pack(anchor = 'w')
self.checkbutton3.pack(anchor = 'w')
self.checkbutton4.pack(anchor = 'w')

# Criando os botões
self.botao = Button(self.frame_baixo,
text='Exibe',command=self.exibe)
self.botao_sair = Button(self.frame_baixo,
text='Sair',command=self.janela_principal.quit)

# Empacotando os botões
self.botao.pack(side='left')
self.botao_sair.pack(side='left')

# Empacotando os frames na janela principal
self.frame_cima.pack()
self.frame_baixo.pack()

# Rodando
mainloop()

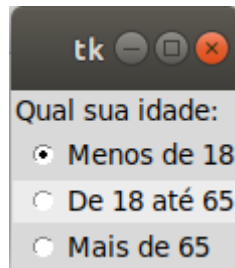
def exibe(self):
    self.texto = 'Você curte: \n'
    if self.checkvar1.get() == 1:
        self.texto += 'MPB\n'
    if self.checkvar2.get() == 1:
        self.texto += 'Música clássica\n'
    if self.checkvar3.get() == 1:
        self.texto += 'Metal\n'
    if self.checkvar4.get() == 1:
        self.texto = 'Tá de sacanagem né?'
    messagebox.showinfo('Seu gosto musical:', self.texto)

gui = MinhaGUI()

```

- **Quando usar Radiobutton ou Checkbutton**

Você deve usar *radiobutton* quando quiser que o usuário selecione uma, e posso selecionar apenas uma, opção. Por exemplo:



Não tem como você ter menos de 18 e entre 18 até 65 ao mesmo tempo. Nem tem como ter de 18 até 65 e mais de 65 ao mesmo tempo. Nesse caso, use sempre botões de rádio.

Agora se o usuário puder escolher mais de uma opção, como gosto musical, aí sim use *checkbutton*.

Tratamento de Eventos

Neste tutorial de GUI, vamos aprender o que são eventos, para que servem, como funcionam bem como fazer seu tratamento (*event handling*) em programação gráfica no Python.

- **Tratamento de Eventos - Event Handling**

Dizemos que a programação gráfica é *event driven*, ou seja, acionada por eventos.

Esses eventos nada mais são que algum tipo de interação da GUI com o usuário, como digitar algum texto, clicar no mouse, fechar uma janela, maximizar, selecionar algum radiobutton etc.

Os programas em GUI são ditos assíncronos, pois eles rodam e ficam lá parados, esperando o usuário fazer algo, diferente de um script puramente textual, que segue uma ordem mais direta e clara.

Em GUI, algo só acontece se você fizer algo acontecer.

E quando você faz algo, algum evento, uma informação, um evento é enviado pro programa, através de um objeto da classe **Event**.

Para processar ou tratar um evento, precisamos de duas coisas:

bind: o programa tem que ligar, vincular (*bind* em inglês) um evento a um componente gráfico, um widget

event handler (ou callback): temos que implementar um 'tratador' de eventos

Ou seja, temos que fazer uma ligação de um evento com um componente, por exemplo, clicar num botão, e criar uma maneira de processar esse evento: quando clicar no botão, abra outra janela, por exemplo.

Em outras palavras, o *event handler* nada mais é que um bom e velho conhecido método que é sempre invocado quando um evento ocorre.

- **Exemplo de Event Handling: bind() e <Button-1>**

Para fazermos uma *bind* num widget, usamos o método `bind()`:

- `widget.bind(evento , tratador_do_evento)`

Um exemplo de evento é o "**<Button-1>**"

Ele fica 'de olho' se o primeiro botão do mouse (o mais da esquerda) é pressionado.

Vamos dar um exemplo com o *widget* do tipo **Frame**.

Criamos uma janela principal, que é um objeto da classe **Tk()**

Em seguida, criamos um frame nessa janela principal, de tamanho 100x100

E associamos ao widget *meuframe* o bind:

`meuframe.bind("<Button-1>", tratador_evento)`

Pronto. Sempre que alguém clicar no *widget* o evento **<Button-1>** ocorre e chamamos o método **tratador_evento**, enviando pra ele um objeto **event**.

Este método simplesmente printa no terminal de comando o local que foi printado. Essa informação está nos atributos **event.x** (posição horizontal) e **event.y** (posição vertical).

Teste o código:

```
from tkinter import *
```

```
# É o event handler
```

```
def tratador_evento(event):
```

```
    print("Você clicou na posição:", event.x, event.y)
```

```
janela_principal = Tk()
```

```
meuframe = Frame(janela_principal, width=100, height=100)
```

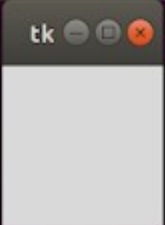
```
meuframe.bind("<Button-1>", tratador_evento)
```

```
meuframe.pack()
```

```
janela_principal.mainloop()
```

O resultado vai ser algo assim:

```
Você clicou na posição: 20 16
Você clicou na posição: 59 43
Você clicou na posição: 93 85
Você clicou na posição: 97 93
Você clicou na posição: 99 98
Você clicou na posição: 97 99
Você clicou na posição: 3 93
Você clicou na posição: 2 2
Você clicou na posição: 1 1
Você clicou na posição: 1 1
```



1. <Button-1> se refere ao botão mais a esquerda do mouse.
2. <Button-2> se refere ao botão do meio, quando existe.
3. <Button-3> é o evento que se refere ao botão mais a direita.

Experimente clicar com o botão mais a direita e veja que nada ocorre.
Troque <Button-1> por <Button-3> e tente novamente.

É como se seu programa GUI ficasse o tempo todo perguntando:

Evento ocorreu? Não? Ok

Evento ocorreu? Não? Tá bom

Evento ocorreu? Não? Tudo certo

....

Evento ocorreu? Sim? Qual? Ah, o <Button-3>, ok, então chama o método **tratador_evento**, ele que vai tratar ele.

...

Evento ocorreu? Não? ...

O Python fica na espreita, aguardando algum evento ocorrer. Se ocorrer, ele identifica qual foi e chama o *handler* correspondente.

E o que é esse handler? E o que ele faz?

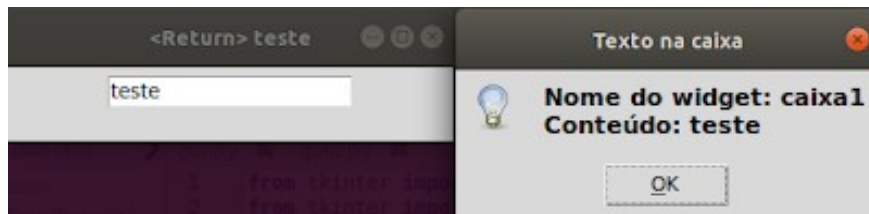
Simples: o que você quiser, o que você decide, é você quem manda nessa bagaça toda. Vamos aprender um pouco mais como tratar alguns eventos.

O evento <Return> : Pressionando a tecla enter

Neste tutorial, vamos aprender como lidar e tratar o evento de pressionar a tecla enter do teclado, através do <Return> event.

- **<Return> event - Pressionar tecla enter**

Sem mais delongas, vamos criar a seguinte aplicação:



- Vamos usar o [widget Entry](#)

Na janelinha principal, exibimos um widget do tipo *Entry* de entrada de dados.

Você digita algo e dá enter.

Ao fazer isso, vai abrir uma nova janela, com o nome do *widget* e o conteúdo que está dentro dele.

Vamos te mostrar o código, e logo abaixo, a explicação:

```
from tkinter import *
from tkinter import messagebox

class EntryTeste( Frame ):
    def __init__( self ):
        # Inicializando configurações do frame
        Frame.__init__(self)
        self.pack()
        self.master.title( "<Return> teste" )
        self.master.geometry( "300x50" )
        self.frame = Frame( self )
        self.frame.pack( pady = 5 )

        # Criando a caixa de entrada
        self.caixa = Entry( self.frame, name = "caixa1" )

        # Fazendo o bind dela com o event handler exhibe()
        self.caixa.bind( "<Return>", self.exibe )
```

```

self.caixa.pack( side = LEFT, padx = 5 )

# Looping pra rodar a GUI
self.mainloop()

# Event handler
def exibe(self, event):
    nomeCaixa = event.widget.winfo_name()
    conteudoCaixa = event.widget.get()
    messagebox.showinfo( "Texto na caixa", "Nome do widget: " +
nomeCaixa + "\nConteúdo: " + conteudoCaixa )

# Criando um objeto
minhaGui = EntryTeste()

```

A classe onde toda mágica vai acontecer se chama **EntryTeste()** e ela herda a classe `Frame`.

Ou seja, ela é um frame também, ela vai ser a janela principal.

Iniciamos o método construtor chamando o construtor da classe base, e em seguida se empacotando.

Com o método **master.title(string)** definimos o título da janelinha que vai abrir.

Com o método **master.geometry("largura x comprimento")** definimos o tamanho do frame.

Depois, criamos o frame *frame1* que vai ser o container da caixa de entrada, e empacotamos ele com o espaçamento vertical (*pad y*) de 5 pixels.

Vamos criar a caixa de entrada, objeto de nome **caixa** do tipo *Entry*, vamos dar o nome de *caixa1* pra esse widget.

Agora vem a parte mais importante, o **bind**.

Vamos ligar o evento **<Return>**, que representa o ato de apertar o enter com o *event handler* de nome **exibe**, que é um método que será chamado sempre que você der enter.

Esse método recebe o objeto **event**.

Vamos extrair algumas informações interessantes desse objeto.

Vamos usar: **event.widget.winfo_name()** pra pegar o nome do widget
E também: **event.widget.get()** pra pegar o conteúdo que foi digitado na caixa de entrada.

Em seguida, esse método, que é nosso *event handler* (tratador de eventos), vai exibir uma caixa de diálogo com essas duas informações: nome do widget e conteúdo que você escreveu nele, antes de dar enter.

Conseguiu entender?

- **Quebre a cabeça**

Não tenha vergonha de ter que ler, reler, quebrar a cabeça, mexer uma coisa aqui, alterar outra ali.

Faz parte do aprendizado.

Rode o código a seguir, e tente entender linha por linha o que está acontecendo.

Se tiver dúvida de algo, não existe em pesquisar no Google, todo programador faz isso, sempre, pro resto da vida, mesmo os melhores do mundo.

```
from tkinter import *
from tkinter import messagebox

class EntryTeste( Frame ):
    def __init__( self ):
        # Inicializando os frames, títulos e tamanhos
        Frame.__init__( self )
        self.pack( expand = YES, fill = BOTH )
        self.master.title( "Componentes de entrada" )
        self.master.geometry( "325x100" ) # width x length

        # Criando o primeiro frame que vai armazenar
        # as duas primeiras caixas de entrada
        self.frame1 = Frame( self )
        self.frame1.pack( pady = 5 )

        # Criando a primeira caixa de entrada
        self.text1 = Entry( self.frame1, name = "caixa1" )
```

```
# Associando o evento <Return> com o
# event handler exhibe()
self.text1.bind( "<Return>", self.exibe )
self.text1.pack( side = LEFT, padx = 5 )
```

```
# Criando a segunda caixa e seu bind
self.text2 = Entry( self.frame1, name = "caixa2" )
self.text2.insert( INSERT, "Digite um texto aqui" )
self.text2.bind( "<Return>", self.exibe )
self.text2.pack( side = LEFT, padx = 5 )
```

```
# Segundo frame, com as duas outras caixas
self.frame2 = Frame( self )
self.frame2.pack( pady = 5 )
```

```
# Criando a caixa que não dá pra editar
self.text3 = Entry( self.frame2, name = "caixa3" )
self.text3.insert( INSERT, "Texto ineditável" )
self.text3.config( state = DISABLED )
self.text3.bind( "<Return>", self.exibe )
self.text3.pack( side = LEFT, padx = 5 )
```

```
# Criando a caixa estilo senha
self.text4 = Entry( self.frame2, name = "caixa4", show = "*" )
self.text4.insert( INSERT, "Texto escondido" )
self.text4.bind( "<Return>", self.exibe )
self.text4.pack( side = LEFT, padx = 5 )
```

```
mainloop()
```

```
def exhibe( self, event ):
```

```
    # Pegando o nome do widget
    nomeCaixa = event.widget.winfo_name()
    # Pegando o conteúdo do widget
    conteudoCaixa = event.widget.get()
    messagebox.showinfo( "Texto na caixa", nomeCaixa + ": " +
    conteudoCaixa )
```

```
# Chamando a classe
EntryTeste()
```

Eventos de Botão: <Enter> e <Leave>

Neste tutorial de Python, vamos aprender como usar dois eventos associados aos botões: o <Enter> e o <Leave>, e vamos ver como tratar *events* com mouse do usuário.

- **Eventos <Enter> e <Leave> de botões**

Agora que já aprendemos a trabalhar com o widget Button, vamos aprender a usar eventos com botões.

Vamos usar dois eventos:

- <Enter> : Quando você passa o mouse por cima do botão
- <Leave>: Quando você tira o mouse de cima do botão

Vamos fazer dois *binds*:

1. Com o evento <Enter>, que chama o método **passou_por_cima**
2. Com o evento <Leave>, que chama o método **saiu_de_cima**

Ou seja, quando você passa o mouse por cima do botão, um *event handler* é acionado.

Quando você tira o mouse de cima, outro *event handler* é acionado.

Não devemos esquecer de passar o objeto **event** para estes métodos de tratamento, é lá que vamos usar o método `widget.config()` para alterar o relevo do botão.

Esses métodos de tratamento de evento vão simplesmente mudar a textura do botão.

Quando passa o mouse por cima, temos o estilo **GROOVE** (de entrar, simulando o botão entrando).

Quando tiramos o mouse de cima, acionamos o estilo **RAISED** (de levantar o botão que tava pra dentro).

E quando apertamos o botão, o método **apertou** é acionado.

Veja como fica nosso código:

```

from tkinter import *
from tkinter import messagebox

class ButtonEvent( Frame ):
    def __init__( self ):
        Frame.__init__( self )
        self.pack()
        self.master.title( "Botão" )
        self.master.geometry("120x40")

        # Criando botão
        self.botao = Button( self, text = "Clique aqui", command =
self.apertou )
        self.botao.bind( "<Enter>", self.passou_por_cima )
        self.botao.bind( "<Leave>", self.saiu_de_cima)
        self.botao.pack( side = LEFT, padx = 5, pady = 5 )

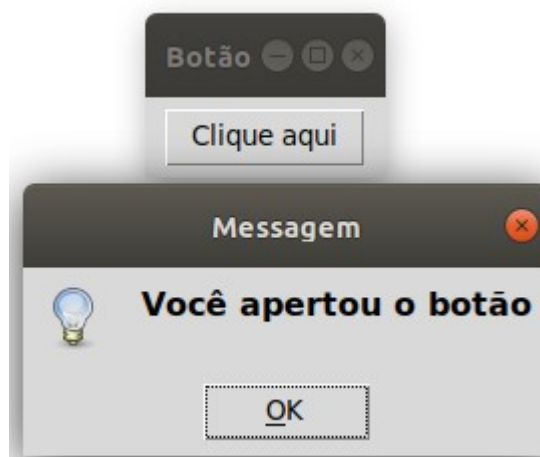
        mainloop()

    def apertou( self ):
        messagebox.showinfo( "Mensagem", "Você apertou o botão" )
    def passou_por_cima( self, event ):
        event.widget.config( relief = GROOVE )
    def saiu_de_cima( self, event ):
        event.widget.config( relief = RAISED )

# Chamando a classe
ButtonEvent()

```

E o resultado:



- **Exercício de GUI**

Crie um programa em GUI que exibe 5 botões.

Quando for criar os objetos do tipo *Button*, passe o parâmetro **relief=ESTILO**, onde ESTILO pode ser:

FLAT

RAISED

SUNKEN

GROOVE

RIDGE

E veja a diferença no estilo de botões.

Tratamento de Eventos com Mouse

Neste tutorial, vamos criar um programa para usar, ao mesmo tempo, 5 tratamentos de eventos envolvendo mouse, como clicar, soltar e arrastar.



- **Eventos com Mouse: <ButtonRelease-1> e <B1-Motion>**

Vamos te mostrar como criar um programa que detecta se o mouse está dentro da janela, se está fora, se você clicou em algum ponto, soltou e arrastou, bem como mostra as coordenadas em pixels desses eventos.

Para isso, vamos usar 5 eventos:

1. <Button-1> - Botão 1 do mouse (mais a esquerda) é pressionado
2. <ButtonRelease-1> - Botão 1 é liberado (released)
3. <Enter> - Mouse entrou no frame (janela)
4. <Leave> - Mouse deixou o frame da janela
5. <B1-Motion> - Botão 1 do mouse foi arrastado (clicar e segurar)

Desses cinco, dois são novos: o <ButtonRelease-1> e o <B1-Motion>, mas bem fáceis de entender com a explicação do programa, a seguir.

Vamos lá, inicialmente criamos nossa classe MouseEvent, que herda a Frame, logo é um frame.

Inicializamos a classe master (Frame.__init__), depois empacotamos o frame, definimos um título e seu tamanho.

Depois, vamos criar o objeto **StringVar()**, que vai ser um *label* que dinâmico, que vai ficar variando e mudando de valor (alterando seu texto). Inicialmente, definimos o texto "Mouse fora da janela" para ele.

Depois, criamos nosso Label, com o parâmetro *textvariable* recebendo o objeto *StringVar*.

Agora vamos fazer os *binds*.

O primeiro, <Button-1>, chama o método **botaoPressionado**, que recebe o evento do clique e mostra sua posição através dos atributos **event.x** e **event.y**, no label.

O segundo bind é com o evento <ButtonRelease-1>, que chama o método **botaoLiberado** que faz o mesmo do método anterior, mas mostra onde você soltou o botão do mouse.

Depois vem o <Enter> e o <Leave>, eventos que já estudamos, que identificam se o mouse entrou ou saiu do *widget* em questão, no caso, o *frame*.

Por fim, o bind com o evento <B1-Motion>, que vai identificar até onde o mouse estará sendo arrastado (clicar, segura e se mover).

Veja como ficou nosso código Python:

```
from tkinter import *
```

```
class MouseEvent( Frame ):
```

```
    def __init__( self ):
```

```
        # Criando o frame, título e tamanho
```

```
        Frame.__init__( self )
```

```
        self.pack( expand = YES, fill = BOTH )
```

```
        self.master.title( "Eventos com Mouse" )
```

```
        self.master.geometry( "300x200" )
```

```
        # String com a posição do mouse
```

```
        self.mousePosicao = StringVar()
```

```
        self.mousePosicao.set( "Mouse fora da janela" )
```

```
        # Criando e posicionando o label embaixo
```

```
        self.positionLabel = Label( self, textvariable = self.mousePosicao )
```

```
self.positionLabel.pack( side = BOTTOM )
```

```
# Criando os binds com os eventos de mouse
```

```
self.bind( "<Button-1>", self.botaoPressionado )
```

```
self.bind( "<ButtonRelease-1>", self.botaoLiberado )
```

```
self.bind( "<Enter>", self.entrouJanela )
```

```
self.bind( "<Leave>", self.saiuJanela )
```

```
self.bind( "<B1-Motion>", self.mouseArrastado )
```

```
mainloop()
```

```
def botaoPressionado( self, event ):
```

```
    self.mousePosicao.set( "Pressionado em [ " + str( event.x ) +  
                           ", " + str( event.y ) + " ]" )
```

```
def botaoLiberado( self, event ):
```

```
    self.mousePosicao.set( "Solto em [ " + str( event.x ) +  
                           ", " + str( event.y ) + " ]" )
```

```
def entrouJanela( self, event ):
```

```
    self.mousePosicao.set( "Mouse na janela )
```

```
def saiuJanela( self, event ):
```

```
    self.mousePosicao.set( "Mouse fora da janela" )
```

```
def mouseArrastado( self, event ):
```

```
    self.mousePosicao.set( "Arrastado até [ " + str( event.x ) +  
                           ", " + str( event.y ) + " ]" )
```

```
# Chamando a classe
```

```
MouseEvent()
```

Tratamento de Eventos com Teclas do teclado

Agora que já aprendemos a tratar eventos envolvendo mouse (clique, soltar e arrastar), vamos ver como fazer o *event handling* com seu teclado, quando o usuário pressionar alguma tecla.

- **Eventos de Teclado**

Control+C copia, Ctrl+V cola, Alt+F4 fecha a janela etc etc. Esses são alguns 'atalhos' que conhecemos no dia a dia de quem lida com informática.

Nada mais são que eventos.

O navegador, editor de texto ou qualquer janela que esteja usando, é uma GUI, é uma interface gráfica pro usuário e também responde a eventos, como os de teclado, que vamos estudar agora.

- **Keyboard Events**

<KeyPress> ou <Key> - Dispara quando qualquer tecla é pressionada

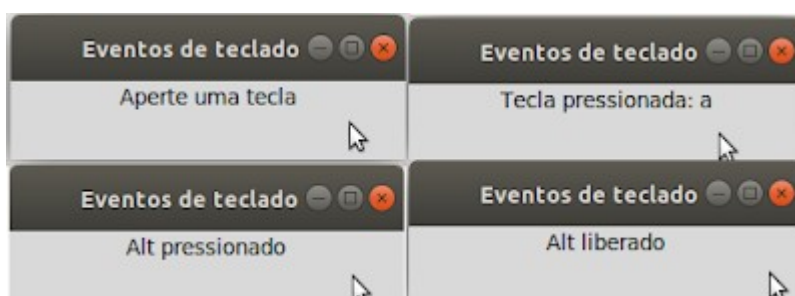
<KeyRelease> - Dispara quando uma tecla é solta ou liberada

<KeyPress-tecla> ou <Key-tecla> - Quando uma tecla específica, a **tecla**, é pressionada

<KeyRelease-tecla> - Quando a **tecla** específica é solta ou liberada

<key> - Abreviação ou atalho, faz o mesmo de <KeyPress-tecla>

<prefixo-tecla> - Dispara quando você aperta a tecla específica **tecla** enquanto segura a **prefixo**. Esse prefixo pode ser Alt, Shift ou Control apenas.



- **Detectar qual tecla foi pressionada**

Vamos colocar esses eventos em prática?

Vamos criar um programa que aguarda o usuário digitar alguma tecla, e exibe na tela qual foi pressionada e quando foi liberada.

Vamos criar a classe `KeyboardEvent`, que é do tipo `Frame`.

Inicializamos sua classe master, empacotamos, damos um título e definimos um tamanho.

Depois, vamos criar o objeto **mensagem**, do tipo `StringVar()`, que vai ter uma string dentro dela que vai ficar mudando, conforme você pressiona e solta as teclas.

Passamos esse objeto como argumento do parâmetro *textvariable*, de nossa `Label label`.

Inicialmente, o texto em *mensagem* é "Aperte uma tecla", é o que vai aparecer no início. Empacotamos o label.

Agora vamos fazer os binds, associar os eventos ao métodos.

Se apertar qualquer tecla, o evento `<KeyPress>` chama o método *teclaPressionada*, que seta a *mensagem* com uma string mostrando a tecla pressionada, que é armazenada no atributo **event.char**

O mesmo para `<KeyRelease>`, que chama o método *teclaLiberada* quando você solta a tecla.

Para detectarmos teclas como Control, Alt e Shift, temos que usar *events* específicos.

Por exemplo, para detectar a tecla Alt da esquerda (Left, L), usamos: `<KeyPress-Alt_L>` e para detectar quando ela foi liberada, usamos `<KeyRelease-Alt_L>`

Veja como ficou nosso código Python:

```
from tkinter import *
from tkinter import messagebox

class KeyboardEvent(Frame):
    def __init__(self):
```

```

Frame.__init__(self)
self.pack(expand=YES, fill=BOTH)
self.master.title("Eventos de teclado")
self.master.geometry("250x50")

# Label e Stringvar que vao exibir a tecla
self.mensagem = StringVar()
self.label = Label(self, textvariable = self.mensagem)
self.mensagem.set("Aperte uma tecla" )
self.label.pack()

# Fazendo os binding no frame
self.master.bind( "<KeyPress>", self.teclaPressionada )
self.master.bind( "<KeyRelease>", self.teclaLiberada )
self.master.bind( "<KeyPress-Alt_L>", self.altPressionado )
self.master.bind( "<KeyRelease-Alt_L>", self.altLiberado )

mainloop()

def teclaPressionada( self, event ):
    self.mensagem.set( "Tecla pressionada: " + event.char )
def teclaLiberada( self, event ):
    self.mensagem.set( "Tecla solta: " + event.char )
def altPressionado( self, event ):
    self.mensagem.set( "Alt pressionado" )
def altLiberado( self, event ):
    self.mensagem.set( "Alt liberado" )

# Chamando a classe
KeyboardEvent()

```

Gerenciadores de Layout: pack, grid e place

Neste tutorial de programação gráfica em Python, vamos aprender melhor como gerenciar o layout de nossos programas de interface gráfica.

•Gerenciadores de Layout

Como já aprendemos a usar botões, caixa de entrada, caixa de diálogo, radio e checkbuttons, bem como tratar diversos eventos, chegou a hora de aprender como organizar melhor esse amontado de widgets!

Você deve ter notado que fomos adicionando botões, labels...e eles foram se encaixando em um canto, em outro...se você é dos que gosta de fuçar, certamente mexeu e viu widgets sumirem, ou ficarem desalinhados, feios etc.

Os gerenciadores de layout servem para definirmos melhor o posicionamento de nossos widgets em containers (janelas Tk() ou Frame() por exemplo), de modo a organizar e deixar eles esteticamente melhores.

Existem três principais gerenciadores de layout:

1. pack - vai adicionando os widgets na ordem que forem sendo empacotados
2. grid - adiciona através de posições de linhas e coluna, como uma tabela ou grade (grid)
3. place - adiciona widgets em posições específicas, com tamanhos específicos

• Gerenciador de Layout: **pack**

O gerenciador pack é o que vínhamos usando, no decorrer de nossos tutoriais de programação gráfica em Python, e é o mais básico.

Por padrão, ele vai adicionando os widgets na ordem quem forem empacotados (chamando o método pack()), de cima pra baixo.

Porém, existem várias opções para empacotar os componentes:

- side - indica de que lado você quer que seja adicionado (TOP, BOTTOM, LEFT e RIGHT). Se quer um alinhamento horizontal, use side=LEFT, por exemplo

- fill - para preencher um espaço. Pode usar como argumento X, Y ou BOTH, e ele vai cobrir todo o espaçamento horizontal (X), vertical (Y) ou ambos (BOTH)
- expand - pode passar como YES ou NO, para definir se o widget vai preencher todo o espaço extra do container ou não.
- padx e pady - preenchem o ao redor do widget com espaço em branco
- pack_forget - retira um widget empacotado do container que foi inserido anteriormente

Estude o seguinte código, rode o script e veja o resultado você mesmo:

```
from tkinter import *
```

```
class PackDemo( Frame ):
    def __init__(self):
        # Inicializando o frame
        Frame.__init__(self)
        self.master.title("Gerenciador pack")
        self.master.geometry( "600x250" )
        self.pack( expand = YES, fill = BOTH)

        # Botao que adiciona outros, no topo
        self.botao1 = Button(self, text = "Adicionar Botão",
                             command = self.adicionarBotao)
        self.botao1.pack(side = TOP)

        # Botão 2, no fundo e preenche X e Y
        self.botao2 = Button(self,
                             text = "expand = NO, fill = BOTH")
        self.botao2.pack( side = BOTTOM, fill = BOTH )

        # Botão3: na esquerda, fill só na horizontal
        self.botao3 = Button( self, text = "expand = YES, fill = X" )
        self.botao3.pack( side = LEFT, expand = YES, fill = X )

        # Botão4: na direita, fill só na vertical
        self.botao4 = Button( self, text = "expand = YES, fill = Y" )
        self.botao4.pack( side = RIGHT, expand = YES, fill = Y )

        mainloop()

    def adicionarBotao( self ):
        Button(self, text = "Botão novo" ).pack(pady = 5)
PackDemo()
```

- **Gerenciador de Layout: grid**

Como o nome sugere, ele vai dividir o container em uma grade (grid em inglês), assim podemos posicionar os elementos dividindo o Frame, por exemplo, em x linhas e y linhas.

Cada posição é chamada de célula, ou seja, cada célula é composta por um valor de linha, um valor de coluna e pode ter um widget

A primeira linha (row) é a 0, a segunda coluna é a de número 1, a terceira é de número 2...

A primeira coluna (column) é a 0, a segunda é a 1, a terceira é a 2...

Definimos a posição de cada widget com o método **grid()**, que recebe os parâmetros *row* e *column*, com a respectiva posição e numeração de cada linha e coluna.

Vamos, por exemplo, criar 9 labels, e exibir eles numa *grid* três por três, ou seja, 3 linhas e 3 colunas.

O código é o seguinte:

```
from tkinter import *
```

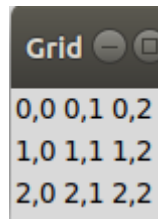
```
class Grid( Frame ):
    def __init__(self):
        # Inicializando o frame
        Frame.__init__(self)
        self.master.title("Grid")
        self.master.geometry( "80x80" )

        # Criando os labels e adicionando
        # com o método grid()
        for linha in range(3):
            for coluna in range(3):
                Label(self.master, text=str(linha)
                    +',' +str(coluna)).grid(row=linha,column=coluna)

        mainloop()
```

```
Grid()
```

E o resultado:



Veja, o primeiro label é o da posição 0,1

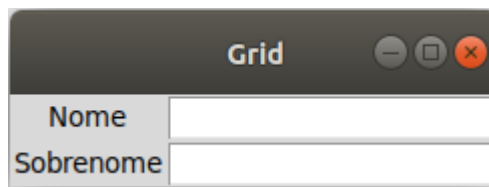
O segundo, na primeira linha e segunda coluna é 0,1

Na primeira linha e terceira coluna temos o 0,2

E assim vai...tente ver uma 'tabela', uma 'grade' ao usar o grid.

Sempre que quiser dispor seus elementos, alinhados por linhas e colunas, use o grid.

Um exemplo comum de programas que usam, são os que tem algum formulário, veja:



Veja como fica bem alinhado e bonitinho.

O código pra fazer isso é:

```
from tkinter import *
class Grid( Frame ):
    def __init__(self):
        # Inicializando o frame
        Frame.__init__(self)
        self.master.title("Grid")

        Label(self.master, text="Nome").grid(row=0)
        Label(self.master, text="Sobrenome").grid(row=1)

        entrada1 = Entry(self.master)
        entrada2 = Entry(self.master)

        entrada1.grid(row=0, column=1)
        entrada2.grid(row=1, column=1)

        mainloop()
```

Grid()

Note que quando omitimos alguma opção *row* ou *column*, é porque o valor padrão é 0.

Para se aprofundar melhor no *manager* grid, sugerimos pesquisar mais sobre a opção *sticky*, os métodos *rowconfigure* e *columnconfigure*.

- **Gerenciador de Layout: *place***

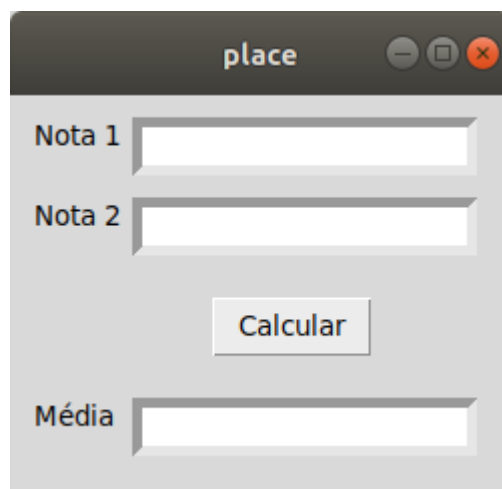
Esse gerenciador não é tão usado, mas é simples de usar, pois basta dizer onde quer inserir cada widget. Pronto.

Porém, isso complica um pouco as coisas.

Note que o *pack* e o *grid* faz muita coisa por você, automaticamente, você se estressa com pouca coisa.

O *place* é mais 'rigoroso', digamos assim, você precisa 'cravar' locais, saber exatamente onde vai cada coisa, até onde, até que altura, até que largura etc. Não vamos nos aprofundar nele.

Para fazer um programa com essa cara:



Rode este código Python:

```
from tkinter import *  
  
class Place( Frame ):  
    def __init__(self):  
        top = Tk()  
        top.title("place")
```

```
top.geometry("250x200")
```

```
# Label e primeira entrada
```

```
label1 = Label(top, text = "Nota 1")
```

```
label1.place(x = 10, y = 10)
```

```
entrada1 = Entry(top, bd = 5)
```

```
entrada1.place(x = 60, y = 10)
```

```
# Label e segunda entrada
```

```
label2 = Label(top, text = "Nota 2")
```

```
label2.place(x = 10, y = 50)
```

```
entrada2 = Entry(top, bd = 5)
```

```
entrada2.place(x = 60, y = 50)
```

```
# Label e terceira entrada
```

```
label3 = Label(top, text = "Média")
```

```
label3.place(x = 10, y = 150)
```

```
entrada3 = Entry(top, bd = 5)
```

```
entrada3.place(x = 60, y = 150)
```

```
# Botão
```

```
botao = Button(top, text = "Calcular")
```

```
botao.place(x = 100, y = 100)
```

```
top.mainloop()
```

Place()



```
def add5(x): def dotwrite(ast): label=symbol.sym_name.get(int(ast[0]),ast[0])  
return x+5     nodename = getNodeName() %s [label="%s" % (nodename, label),  
print '= %s'];  
else:         if isinstance(ast[1], str):  
print ''      if ast[1].strip(): print '%s' % name,  
for n, child in enumerate(ast[1:]):  
children.append(dotwrite(child))
```

Certificado Curso Python Progressivo

