**Aula 1**

**Parte 1**

Olá pessoal, hoje iniciaremos nosso curso “Fundamentos de Linguagem de Programação em Python”. Acredito que todos já baixaram o Anaconda e junto com ele o Jupyter, que será nosso ambiente de desenvolvimento, em inglês IDE. Essa será uma aula introdutória aonde iremos explorar um pouco nossas ferramentas de estudos.

Antes deixa eu explicar como será o curso, toda segunda-feira eu colocarei aqui nesse canal vídeos, de até 1 hora, explicando o conteúdo da semana e alguns exemplos práticos de como usar o conteúdo em alguns exercícios. O vídeo será postado toda segunda-feira as 14 horas, entre as 13-15 horas irei colocar no classroom uma sessão aonde vocês irão deixar suas dúvidas e colocarão as atividades, somente quando terminadas. Explicarei melhor no vídeo de segunda já com a lista. Combinaremos um horário na quarta-feira, um horário de 2 horas, aonde eu responderei as dúvidas de vocês, mas podem deixar as dúvidas de segunda até sexta. Na sexta sairá mais um vídeo de até 1 hora, agora com a resolução da lista e respondendo algumas dúvidas que achei interessante colocar no vídeo.

Primeiro, antes de tudo, vamos relembrar de como baixar o material didático desse curso. Na página inicial do GitHub, no canto superior direito, temos uma barra de pesquisa, escrevemos “psloliveirajr/” e pesquisamos. Nós queremos abrir o repositório “Introdução\_a\_Python3”. Vamos em “Code” e clicamos na opção “Download ZIP”, finalizando o download o arquivo vai para sua pasta de downloads e extrairemos o arquivo. Podemos copiar essa pasta que surgiu para uma outra pasta se quiser, eu normalmente coloco em uma pasta nos meus documentos deixando meus arquivos mais organizados. Periodicamente eu irei atualizar esse repositório do GitHub, com listas ou outras modificações no material, então vocês terão que refazer esse procedimento algumas vezes. Mas não se preocupem, que irei informar quando necessário.

Já com os arquivos no computador, começaremos a dar uma olhada no Jupyter. Para abrir o programa, basta ir na barra de pesquisa do Windows e escrever “Jupyter Notebook” e abrir. No meu caso abriu automaticamente no Google Chrome, caso não abra, clica aqui no ícone do Jupyter na barra de tarefas e copia qualquer uma das URLs e cola na barra de endereço do seu navegador. Basta navegar no seu computador aonde está os arquivos do curso, achando a pasta, é só abrir o arquivo “Curso.ipynb”. É esse material aqui que usaremos no nosso curso, mas vamos olhar algumas funções do Jupyter.

Esse tipo de arquivo, normalmente chamado de notebook, é composto por células aonde os códigos e textos são escritos. Células com “In [ ]” são células de códigos, aqui em cima podemos ver que está na opção “Code”, e células sem o “In” do lado são células de texto, aqui em cima podemos ver que está na opção “Markdown”. Podemos alterar o momento que quiser essas células. Para rodar os códigos basta selecionar a célula de código e clicar em “Run”, quando o código termina de rodar um número é adicionado no “In” e teremos um “Out”. Podemos adicionar células nova no botão “+”, podemos cortar células, podemos copiar células, colar células e mover as células para cima ou para baixo.

Além do Jupyter, também faremos o uso de outros ambientes no curso, como Google Colab. Ele é super parecido com os Jupyter e tem muitas outras funcionalidades, o ponto mais importante de o Colab é que ele roda na nuvem, ou seja, zero custo do seu computador quando rodar os programas. Primeiro vamos abrir o curso nele. Nós podemos abrir o material nele sem que seja necessário baixar todo o material lá do GitHub, mas para isso vai precisar de ter uma conta no GitHub, basta vir aqui em “Arquivos, “Abrir Notebook”, “GitHub”, ele irá pedir uma autorização para vincular as contas, voltemos lá no GitHub e copiamos o URL do nosso repositório, finalmente abrimos o material. Para rodar os códigos aqui, basta clicar no botão de play, podemos adicionar células de texto ou de código, mover, apagar, colar, copiar. Como abrimos o material através do GitHub, temos que criar uma cópia do arquivo no seu drive caso queira salvar, e ela fica salva em uma pasta dessa no seu drive.

Os exercícios serão feitos em arquivos como esse, abertos no Colab, porque existe a opção de compartilhamento que se compartilhado comigo eu posso entrar na sua lista e fazer comentários em alguma dúvida que vocês tenham e comuniquem comigo, e com isso eu quero fazer uma proposta com vocês. Eu posso criar um arquivo com os exercícios para cada um de vocês e compartilhar esse arquivo, um para cada um, daria um trabalho bem grande, mas pode ser feito. Ou então vocês fazem esse mesmo procedimento que fizemos para abrir o material do curso, mas, em vez disso, abririam a lista em branco que deixarei no GitHuh. Eai vocês fazem uma cópia para vocês dentro do drive de vocês, coloquem o nome e segundo nome no título da lista junto com o número da lista, desta forma “PrimeiroSeguno\_lista1. Depois vocês irão compartilhar esse documento comigo colocando meu e-mail e me enviando, com a lista em mãos eu falo melhor sobre como tirar as dúvidas.

**Programa**

Vamos falar um pouco sobre programação e Python3. Neste curso aprenderemos como programar em Python, isso significa, iremos criar programas escritos na linguagem Python. Então vamos entender oque é um programa.

“Um programa é uma sequência de instruções que especifica como executar uma computação. A computação pode ser algo matemático, como resolver um sistema de equações ou encontrar as raízes de um polinômio, mas também pode ser uma computação simbólica, como pesquisar e substituir texto em um documento ou em algo gráfico, como processamento de uma imagem ou reproduzindo um vídeo.”

Nossa atenção neste curso será em problemas relativamente simples, como, por exemplo, resolver equações e modificar texto. Trabalhos mais complexos, deixarei para aqueles que queiram se aventurar no mundo dos computadores.

Um programa é composto por Algoritmos e Estruturas de dados.

“Um **algoritmo** é a descrição de um padrão de comportamento, especificado em termos de um conjunto bem definido e finito de ações primitivas que podem ser executadas. Já **estruturas de dados** é um modo particular de armazenamento e organização de dados em um computador de como que possam ser usados eficientemente.”

Podemos pensar um algoritmo como se fosse uma receita para fazer uma lasanha, aonde você tem um conjunto de instruções que devem ser seguidas de acordo com a sequência que é apresentada. Algumas instruções não são executadas imediatamente, precisamos repetir um conjunto de instruções antes de executar a próxima, como, por exemplo, na montagem precisamos fazer várias camadas na lasanha até colocar o queijo parmesão. Outras instruções podem não ser mencionadas, como, por exemplo, corta a cenoura antes de colocar na panela. Também podemos mudar a ordem de algumas instruções, como, por exemplo, fazer o molho branco e depois o molho Bolonhesa.

Já a estrutura de dados vai depender da linguagem de programação que utilizarmos, no nosso caso será o Python.

Lembre-se que um bom programa é necessita de um bom algoritmo e uma boa estrutura dos dados.

**Programação Estruturada**

“Metodologia de projeto de programas que visa:

1. Facilitar o desenvolvimento dos programas
2. Facilitar a leitura (entendimento) dos programas
3. Permitir a validação a priori dos programas
4. Facilitar a manutenção e modificação dos programas”

Esses são os cuidados que vocês devem tomar quando forem construir um programa, isso ajuda vocês e a mim quando for corrigir as atividades.

**Ciclo de vida de um Software**

Aqui vai mais uma dica que vocês devem seguir para ter solucionarem seus problemas.

Nós temos um problema e queremos uma solução, então siga esses passos que a vida de vocês será bem tranquila.

**1. Especificar os requisitos do problema**

- Preparar uma especificação completa e não ambígua.

Normalmente esses problemas podem ser especificados pelo seu professor, pelo seu orientador. Então essas perguntas tem que vir de forma clara.

**2. Analisar o problema**

- Entender o problema.

- Avaliar soluções alternativas.

- Escolher soluções mais adequadas.

O problema pode ser oque já falamos, fazer uma lasanha. Para isso você tem que saber oque é uma lasanha e ter uma ideia de como faz ela

**3. Projetar o programa para solucionar o problema**

- Fazer o projeto de cima para baixo (top-down) do sistema.

- Para cada módulo, identificar as principais estruturas de dados e subprogramas associados.

- Desenvolver algoritmos e estruturas de dados dos subprogramas.

No algoritmo da lasanha, temos 4 subprogramas, fazer a massa da lasanha, o molho bolonhesa, o molho branco e a montagem. Cada um dos subprogramas é desenvolvido com os próprios algoritmos.

**4. Implementar o projeto**

- Codificar a solução.

- Corrigir erros de codificação.

**5. Testar e validar o programa**

- Testar o código e validá-lo, se correto.

Você vai ter que comer a lasanha pra ver se ela ficou boa, no caso podemos fazer uma lasanha pequena antes para depois fazer uma maior.

**6. Manter e atualizar o programa**

- Executar o sistema;

- Avaliar seu desempenho;

- Remover novos erros identificados, assim que detectados;

- Realizar modificações de forma a manter o sistema atualizado;

- Validar as modificações.

Essa parte é mais voltadas para programas mais avançados, porém é basicamente o mesmo para o que vamos fazer.

**Características da linguagem Python 3.x**

**Linguagem de programação de alto nível**

Não se assuste não é que ela seja mais difícil é o contrário, quanto mais o nível mais fácil é. Isso porque o nível de abstração é mais elevado, a linguagem é mais próxima da linguagem humana que utilizamos.

**Multiparadigma (procedural, funcional e orientado a objetos)**

A linguagem Python é multiparadigma, porque ela pode ser procedural, funcional e orientado a objetos tudo ao mesmo tempo ou só duas ou só umas delas. Ser procedural é ser basicamente uma linguagem que segue passos. Ser funcional é usar funções da própria linguagem ou criadas, iremos aprender sobre essas funções. Não irei falar de orientado a objetos, pois não iremos aprender nesse curso.

**Multiplataforma (Windows, Linux, iOS, etc.)**

Ser multiplataforma quer dizer que um código feito em Windows pode rodar no Linux, no iOS ou em qualquer outro sistema operacional.

**Interpretada (padrão ou interativo)**

Ser uma linguagem interpretada é oque torna o Python mais simples, porque não precisamos nos preocupar com alguns processos que os programas, que utilizaremos, irão fazer para seu código seja entendido pelo seu computador. Isso nos permite executar o Python através de Scripts, que são um conjunto de linhas de códigos que executarão varias atividades, nós iremos nos referir a um script um programa completo. Também podemos trabalhar com o Python interativamente, que é basicamente o que faremos neste material, rodar pequenas linhas de códigos uma por uma.

O Jupyter eu posso criar uma célula com varias linhas de código, formando um script ou só rodar uma linha de código, então ele é basicamente os dois formatos de Python em um só. No Anaconda temos alguns programas bem específicos para trabalhar de forma interativas, por exemplo, o QT Console. Podemos rodar pequenas linhas de comando sempre precisar rodar um código inteiro.

**Gerência de memória automatizada (coletor de lixo)**

Programas escritos em linguagens como o C, quando rodados, definem a quantidade de memória ram que o programa irá utilizar do computador já quando começa a rodar, no Python a quantidade de memória ram vai variando conforme vamos utilizando o programa.

Entender isso é fundamental para aqueles que querem avançar na programação. Para aqueles que não sabem, deixa eu falar sobre memória ram.

Um computador tem um máximo de memória ram, seja a que você escolheu quando montou ou que já era definida quando comprou o computador. Podemos ver a quantidade dessa memoria indo no “Gerenciador de tarefas do Windows”. Apertamos em “Mais detalhes” e vamos em “Desempenho”, aqui podemos ver o consumo de memória do computador. Processos como o Chrome gastam bastante memória ram, o Jupyter aberto no seu PC também, por isso recomendo aqueles que tiverem computadores mais fracos usarem o Coolab.

Na próxima aula falaremos do processo de armazenamento dos dados, agora vamos ver um pouco sobre esses dados em Python.

**Objetos**

“Todos dados em um programa Python são representados por objetos e pelas relações entre objetos. Cada objeto no Python possui três características:

* **Tipo (classe)**: números, string, lista,etc.
* **Valor**: Valores podem ser números, palavras, etc.
* **Identidade**: Parte da memória do computador que o objeto ocupa “

**A hierarquia conceitual do Python**

De uma perspectiva mais concreta, os programas Python podem ser decompostos em módulos, instruções, expressões e objetos, da seguinte maneira:

1. Os programas são compostos de módulos.
2. Módulos contêm estruturas.
3. Estruturas contêm expressões.
4. Expressões criam e processam objetos.

**Aula 1**

Olá pessoal, hoje começamos a semana 2 do nosso curso, “Fundamentos de Linguagem de Programação em Python”. Nesta semana iremos estudar sobre estrutura de seleção e funções, como eu já disse, as vezes irei atualizar o material para adequar o conteúdo de forma que vocês consigam entender melhor o conteúdo. Essa semana não foi diferente, fiz umas pequenas modificações, sendo assim irão precisar baixar o material novamente. Basta vir aqui no GitHub e fazer o download do material novamente.

Como eu já tinha pontuado no ultimo vídeo da primeira aula, mas exatamente em “A hierarquia conceitual do Python”, que o Python contém estruturas e essas estruturas contêm expressões e também vimos na semana passada uma dessas estruturas, que foi a estrutura de atribuição. Vimos que essa estrutura cria uma nova variável e fornece a ela um valor. E agora veremos novas estruturas que tem um caráter de seleção. Que pelo nome já podemos deduzir que se tratam de estruturas que irão selecionar tarefas no decorrer do programa, mas como essas estruturas selecionam essas tarefas e quais tarefas elas selecionam???

Essas estruturas utilizam de condições para realizarem suas operações, condições essas que são expressões realizadas entre objetos utilizando de operadores de comparação e/ou operadores booleanos. Eu exemplifiquei o uso desses operadores na semana passada, caso ache necessário volte no vídeo da semana passada aonde eu exemplifico cada um dos operadores. Essas condições sempre nos retornam algum resultado, condições verdadeiras denotamos como **True** e condições falsas denotamos como **False**.

Dependendo do resultado dessas condições essas estruturas irão selecionar tarefas bem especificas. Elas realizaram tarefas dentro da sua própria **Suite**. Não, não são quartos maravilhosos com banheiro embutido que você já sai do banho e deita naquela cama gostosinha depois de um longo dia de estudo/trabalho, não, não é isso. Em Python, um bloco de comandos conteúdo um ou mais comando é chamado e suite. No exemplo abaixo, temos uma suite com cinco comando em sequência.

Vejam que são conjuntos de comandos como já vimos anteriormente, nesse caso essa suite soma e faz o produto entre dois valores fornecidos pelo usuário e imprimi no final o resultado das operações. São suítes como essa que podemos ter dentro de uma estrutura de seleção. Perceba que eu falei **dentro**, vai ser a forma que vou me referir quando falar que algo ocorre em uma suite de alguma estrutura.

A primeira estrutura de seleção é a **estrutura if,** que realiza uma simples execução condicional. O Python é uma linguagem mais próxima da linguagem natural e é escrito com base no inglês, então a tradução de **if** é “E se”. Isso quer dizer que, e se uma condição é verdade faça isso.

Nós montamos a estrutura da seguinte maneira:

* **IF**
* acompanhado da condição
* depois dois pontos
* pula para a próxima linha
* aperta o botão espaço 4 vezes ou aperte o botão **TAB** uma vez, o botão TAB fica normalmente acima do **CAPS LOCK**.
* Nós chamamos esse processo de espaçamento de **IDENTAÇÂO**
* Todos os comandos dentro da estrutura, ou seja, a suite da estrutura fica espaçada esse tanto, o fim da estrutura se dá quando não damos mais o espaçamento

Como eu falei, a condição é uma expressão booleana que usa operadores de comparação e/ou operadores booleanos, resultando **True** ou **False**.

Podemos ver do lado, um diagrama de fluxo, a direção das setas indica as sequências de ações que o programa vai tomar durante seu processamento. Vemos que algumas tarefas são realizadas na suite principal e aí o programa bate no **if** (irei usar bastante a palavra bate nesses casos) e verifica a condição, se a condição resultar em **True**, o programa realiza as operações dentro da suite do if se a condição resultar em **False** o programa ignora todos os comandos da suite e continua o fluxo do programa

EXEMPLOS

Podemos combinar com if a estrutura **else,** formando uma execução alternativa. A tradução mais adequada para o else nesta situação é “se não”, se não isto faça aquilo. O else nunca vem desacompanhado, sempre vem com um if antes dele, pois a **estrutura else** não acompanhada de uma condição, quando a condição do if resultar em False o programa irá ignorar toda a suite do if e rodas a suite do else.

Podemos ver isso melhor no diagrama. O programa está rodando quando ele bate com a condição do if, se resultar True ele roda a suite do if se resultar False ele vai rodar a suite do else. Nunca os dois, ou um ou outro. No final ele volta para a suite principal

EXEMPLO

Nós podemos combinar com o if e o else a **estrutura elif**, formando condicionais encadeado. Não tenho uma boa tradução para essa estrutura, mas ela é uma combinação do else com o if. O elif sempre vem depois de um if e antes de um else. Podemos ver que ele também vem acompanhado de uma condição como o if, mas, também como o else essa condição só testada caso a condição do if resulte em False.

Vejamos no diagrama ao lado. Quando um o programa bate no if ele testa a condição do if, se True ele roda a suite do if e ignora todas as outras, caso o resultado seja false ele irá testar a condição do elif. Caso resulte em True, o programa irá rodar a suite do elif e ignorar todas as outras, é possível colocar mais elif para outras condições e assim tentar contemplar todas as possibilidades de condições para um caso. O programa irá testar o próximo elif somente se o elif anterior resulte em False e assim vai continuar até o ultimo elif em sequência. Caso todos as condições resultem em False o programa ire rodar a suite do else e depois voltando para a suite principal.

EXEMPLO

Condicionais aninhados é uma solução alternativa aos condicionais encadeados

Finalizamos o conteúdo da aula passada falando de funções padrões. Acredito que minha explicação de função fico meio vago quando eu falei, mas não se preocupem que nessa aula iremos adentrar um pouco mais nessa ideia de funções e o benefício do uso dessas funções na programação.

“Em termos simples, uma função é um dispositivo que agrupa um conjunto de instruções para que possam ser executadas mais de uma vez em um programa - um procedimento empacotado chamado pelo nome. As funções também podem calcular um valor de resultado e vamos especificar parâmetros que servem como entradas de função e podem diferir cada vez que o código é executado. Codificar uma operação como uma função a torna uma ferramenta geralmente útil, que podemos usar em vários contextos.”

Começamos estudando a ideia de um bloco de comandos. “Em Python, um bloco de comandos contendo um ou mais comandos é chamado de **suite**. No exemplo abaixo, temos uma suite com cinco comandos em sequência”.

Essa suite pede ao usuário dois valores inteiros, realiza uma operação de soma, uma operação de multiplicação entre os dois valores e no final imprime o resultado dessas duas operações.

Como falei agora pouco, uma função é um procedimento empacotado chamado pelo nome, esses procedimentos ficam dentro de uma suite própria da função, então isso quer dizer que esses procedimentos só podem ser realizados por essa mesma função.

Além daquelas funções básicas, também podemos adicionar novas funções. Sempre é bom fazer novas funções pois, “maximizamos a reutilização de código e minimizando a redundância” e “deixamos o código mais limpo”.

Na imagem podemos ver exemplos de como criar uma função, perceba que eu divido em “Programa Completo”, em “Subprogramas” e “Programa Principal”. Normalmente dizemos que cada função é um sub-programa do programa principal.

A estrutura responsável por criar uma função e atribuir um nome a esta função é a estrutura def. Escrevemos def, espaço, nome da função, abre parêntese e fecha parêntese, uma função pode ter zeros argumentos ou quantos argumentos quiserem. Um argumento pode ser qualquer tipo objeto. Depois dos parênteses colocamos dois pontos e as linhas abaixo é o suite da função, finalizamos a função com a estrutura return. Podemos retorna qualquer tipo de objeto até o objeto do tipo None que é basicamente não retorna nada, normalmente retornamos o resultado das operações feitas dentro da função.

Lembrando as funções padrões, nós temos a função round, que é um objeto do tipo função e usando a função help podemos ver o que a função faz. Ela é uma função que precisa de pelo menos um argumento, o número que quer arredondar, e também podemos dizer o número de casa decimais que queremos que número retorne.

Então quando criarmos uma função podemos definir os argumentos que queremos usar na função. Criei essas funções como exemplo, uma função prod que usa dois argumentos e retorna o produto dos argumentos e uma função soma que retorna a soma dos argumentos.

Lembre-se de nunca esquecer de colocar dois pontos depois de fechar os parênteses e é necessário que toda a suite da função esteja espaçada. Normalmente quando apertamos enter depois de escrever os dois pontos o espaço é dado automaticamente, caso não seja automático aperte TAB e o espaço será dado de forma correta. **Esse procedimento se chama indentação.** Nunca use o botão espaço 4 vezes se não você não vai entrar no Clube do TAB.

<https://thejenkinscomic.files.wordpress.com/2020/04/screenshot-1080.png?w=760>

**Não sei se perceberam, mas sempre nos códigos eu coloco um hastag para adicionar comentários sobre o programa, aconselho a sempre usarem.**

Podemos adicionar as funções uma docstring (um texto explicativo do oque a função faz) que nem a funções padrões tem, basta fazer que nem fiz aqui no exemplo, coloco toda explicação entre aspas triplas. Agora basta rodar esse código que as funções vão estar definidas para serem usadas.

A função prod é um objeto do tipo function, podemos ver sua docstring usando a função help e podemos usar a função no código. Entramos com dois inteiros e ele imprime o retorno da função prod.

A função soma é um objeto do tipo function, podemos ver sua docstring usando a função help e podemos usar a função no código. Entramos com dois inteiros e ele imprime o retorno da soma prod.

**AULA 4**

**Recursividade**

Começaremos hoje falando de recursividade. A recursividade não é uma estrutura e nem nada do tipo, também não é algo próprio da computação, mas em computação a **recursividade** é a definição de um subprograma que pode invocar a si mesmo. Até o momento aprendemos funções como sendo subprogramas, então recursividade aqui vai ser um programa chamando a si mesmo durante o funcionamento do programa principal.

Para entender como isso funciona em Python, vamos primeiro entender escopo e pilha de ativação. Começaremos com o escopo

**Escopo**

“Quando você usa um nome em um programa, o Python cria, altera ou consulta o nome no que é conhecido como namespace - um local onde os nomes residem.”. Por exemplo quando somamos duas variáveis, o Python consulta os objetos e os valores atribuídos a cada variável e depois faz a soma dos objetos (eu falo objeto porque a soma pode ser de duas strings, que nesse caso seria uma concatenação).

“Quando falamos sobre a pesquisa do valor de uma variável em **relação ao código**, o termo **escopo** refere-se a um namespace: ou seja, o local da atribuição de um valor a uma variável no código determina o seu escopo de visibilidade no código.”

“Sendo assim, uma mesma variável pode se referir a diferentes objetos, em um mesmo código, mas essa variável tem que estar em escopos diferentes. Em Python temos o total de 4 tipos de escopos, local (**L**), enclosed function (**E**), global (**G**) e built-in (**B**), formando a regra **LEGB**. Trata-se de uma regra de prioridade para determinar qual escopo o Python irá acessar quando quiser cria, altera ou consulta o nome de uma variável, sendo a maior prioridade o local e menor prioridade built-in”

Podemos imaginar a regra a partir desta imagem, aonde os escopos mais a dentro tem mais prioridades. Podemos ter infinitos escopos do tipo enclosed.

EXEMPLO

**Pilha de ativacao**