**Introdução da aula**



**Qual é o foco da aula?**

Nesta aula, vamos conhecer alguns trechos de código e sua lógica de execução.

**Objetivos gerais de aprendizagem**

Ao longo desta aula, você irá:

* examinar o são estruturas condicionais;
* aplicar as estruturas condicionais em *Python;*
* identificar as estruturas de repetição e suas aplicações

**Situação-problema**

Trabalhar em uma empresa de consultoria focada no desenvolvimento de software é desafiador, pois você pode atender difentes clientes e atuar em diferentes projetos. Embora desafiador, sem dúvida é uma experiência enriquecedora.

Dando continuidade ao seu trabalho na empresa de consultoria de software, o cliente que fabrica peças automotivas requisitou uma nova funcionalidade para o sistema: calcular o imposto de renda a ser deduzido do salário dos colaboradores. O imposto de renda

*"Incide sobre a renda e os proventos de contribuintes residentes no País ou residentes no exterior que recebam rendimentos de fontes no Brasil. Apresenta alíquotas variáveis conforme a renda dos contribuintes, de forma que os de menor renda não sejam alcançados pela tributação."* [(IRPF, 2020)](http://receita.economia.gov.br/acesso-rapido/tributos/irpf-imposto-de-renda-pessoa-fisica" \t "_blank)

Você foi designado para pesquisar a tabela de valores para o imposto de renda do ano de 2020, pensar no algoritmo e implementar a primeira versão da solução. Quando o cliente aprovar, a versão passará para o time de produção. Nessa primeira versão, o programa deve solicitar o salário do colaborador e então, informar qual o valor do imposto que será deduzido do salário.

Já sabe qual ferramenta vai utilizar para implementar a solução? Onde buscar as informações confiáveis sobre o imposto de renda? Como implementar a solução? Faça um bom trabalho e mostre para seu gestor sua evolução!

**Estrutura condicional**



div tabindex="-1" id="notebook" class="border-box-sizing">

Criamos algoritmos para resolver problemas, o que implica tomar decisões. No universo da programação, a técnica que nos permite tomar decisões é chamada de estrutura condicional (MANZANO; OLIVEIRA, 2019).

A figura abaixo ilustra uma cidade e suas vias. Ao dirigir em uma cidade, você precisa decidir quais caminhos (ruas) seguir com o objetivo de chegar a um destino; por exemplo, em um cruzamento, pode-se virar à direita ou à esquerda e/ou seguir em frente.

Um programa funciona de forma similar: deve-se decidir qual caminho será executado em um código. Em geral, em um programa você tem opções de caminhos ou lista de comandos que nada mais são que trechos de códigos que podem ser executados, devendo-se tomar decisões sobre qual trecho de código será executado em um determinado momento.

Fluxo de tomada de decisões de um algoritmo para a escolha de um caminho em uma cidade. Fonte: adaptada de Griffiths e Barry (2009, p. 13).

Para tomarmos decisões, precisamos dos operadores relacionais (quadro abaixo), que já fazem parte do nosso vocabulário, desde as aulas de matemática do colégio. Vamos usá-los para fazer comparações entre objetos (lembrando que em *Python*tudo é objeto, então uma variável também é um objeto).

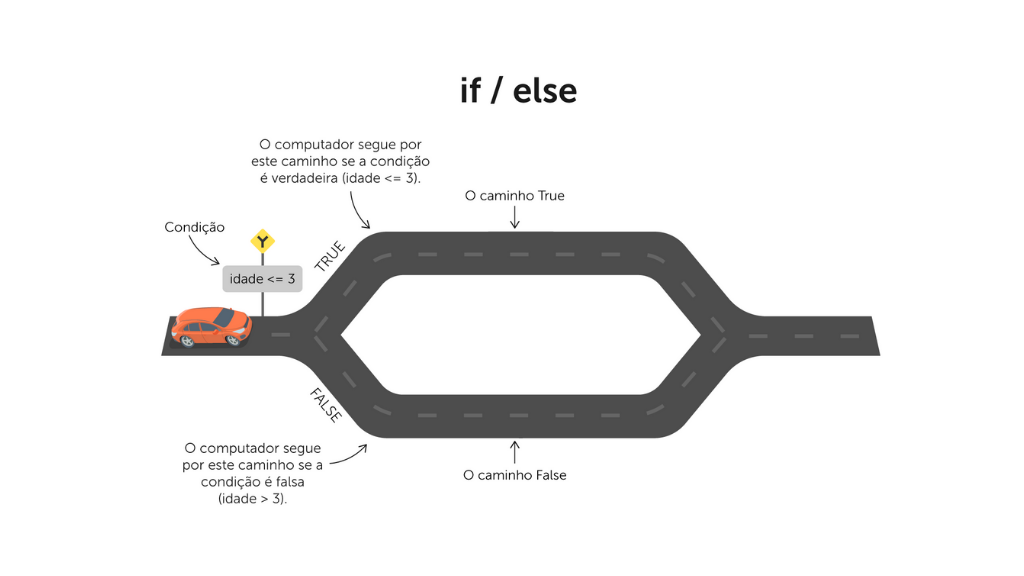
Operadores relacionais. Fonte: adaptado de Docs.Python.

**Estruturas condicionais em Python: If, Elif, Else**



O comando if.. else.. significam se.. senão.. e são usados para construir as estruturas condicionais. Se uma condição for satisfeita, então o fluxo de execução segue por um caminho, se não for satisfeito então segue por outro caminho.

A figura abaixo ilustra um exemplo de tomada de decisão. No exemplo, dois caminhos podem ser seguidos pelo condutor: um verdadeiro (*TRUE*), e um falso (*FALSE*). Assim funciona uma condição: ela pode ser satisfeita (Verdade) ou não (Falsa). No exemplo, se a idade for menor ou igual a 3, o condutor seguirá o caminho verdadeiro. Por sua vez, se ela for maior que 3, seguirá o caminho falso.

Tomada de decisão utilizando Se/senão (If/else). Fonte: adaptada de Griffiths e Barry (2009, p. 13).

Python possui uma sintaxe especial para a construção das estruturas condicionais. Observe o trecho a seguir:

**In[1]**

a **=** 5

b **=** 10

if a **<** b:

    print("a é menor do que b")

    r **=** a **+** b

    print(r)

a é menor do que b  
  
15

Como você pode observar, a construção do if (se) deve ser feita com dois pontos ao final da condição, e o bloco de ações que devem ser feitas. Caso o trecho seja verdadeiro, deve possuir uma indentação específica de uma tabulação ou 4 espaços em branco. Repare que não precisamos escrever if a < b == True.

O código apresentado na entrada 1(In [1]) refere-se a uma estrutura condicional simples, pois só existem ações caso a condição seja verdadeira. Veja como construir uma estrutura composta:

**In[2]**

a **=** 10

b **=** 5

if a **<** b:

    print("a é menor do que b")

    r **=** a **+** b

    print(r)

else:

    print("a é maior do que b")

    r **=** a **-** b

    print(r)

a é maior do que b  
  
5

O código apresentado na entrada 2(In [2]) refere-se a uma estrutura condicional composta, pois existem ações tanto para a condição verdadeira quanto para a falsa. Veja que o else (senão) está no mesmo nível de identação do if, e os blocos de ações com uma tabulação de indentação.

Além das estruturas simples e compostas, existem as estruturas encadeadas. Em diversas linguagens como C, Java, C#, C++, etc, essa estrutura é construída como o comando switch..case. Em *Python*, não existe o [comando](https://docs.python.org/3/tutorial/controlflow.html" \t "_blank)*[switch](https://docs.python.org/3/tutorial/controlflow.html" \t "_blank)*. Para construir uma estrutura encadeada, devemos usar o comando "elif", que é uma abreviação de else if. Veja um exemplo desse tipo de estrutura:

**In [3]**

codigo\_compra **=** 5111

if codigo\_compra **==** 5222:

    print("Compra à vista.")

elif codigo\_compra **==** 5333:

    print("Compra à prazo no boleto.")

elif codigo\_compra **==** 5444:

    print("Compra à prazo no cartão.")

else:

    print("Código não cadastrado")

Código não cadastrado

O código apresentado na entrada 3 (In [3]) mostra a construção de uma estrutura condicional encadeada. Veja que começamos pelo teste com if, depois testamos várias alternativas com o elif e, por fim, colocamos uma mensagem, caso nenhuma opção seja verdadeira, no else.

**Estruturas lógicas em Python: And, Or e Notrograma**



Além dos operadores relacionais, podemos usar os operadores booleanos para construir estruturas de decisões mais complexas.

* Operador booleano and: Esse operador faz a operação lógica E, ou seja, dada a expressão (a and b), o resultado será True, somente quando os dois argumentos forem verdadeiros.
* Operador booleano or: Esse operador faz a operação lógica OU, ou seja, dada a expressão (a or b), o resultado será True, quando pelo menos um dos argumentos for verdadeiro.
* Operador booleano not: Esse operador faz a operação lógica NOT, ou seja, dada a expressão (not a), ele irá inverter o valor do argumento. Portanto, se o argumento for verdadeiro, a operação o transformará em falso e vice-versa.

Observe o exemplo a seguir, que cria uma estrutura condicional usando os operadores booleanos. Um aluno só pode ser aprovado caso ele tenha menos de 5 faltas e média final superior a 7.

**In [4]**

qtde\_faltas **=** int(input("Digite a quantidade de faltas: "))

media\_final **=** float(input("Digite a média final: "))

if qtde\_faltas **<=** 5 and media\_final **>=** 7:

    print("Aluno aprovado!")

else:

    print("Aluno reprovado!")

Digite a quantidade de faltas: 2  
  
Digite a média final: 7  
  
Aluno aprovado!

No exemplo da entrada 4 (In [4]), fizemos a conversão do tipo string para numérico, encadeando funções. Como já sabemos, primeiro são resolvidos os parênteses mais internos, ou seja, na linha 1, será feita primeiro a função input(), em seguida, a entrada digitada será convertida para inteira com a função int(). A mesma lógica se aplica na linha 2, na qual a entrada é convertida para ponto flutuante. Na linha 4 fizemos o teste condicional, primeiro são resolvidos os operadores relacionais, em seguida, os lógicos.

Obs: not tem uma prioridade mais baixa que os operadores relacionais. Portanto, not a == b é interpretado como: not (a == b) e a == not b gera um erro de sintaxe.

Obs2: Assim como as operações matemáticas possuem ordem de precedência, as operações booleanas também têm. Essa prioridade obedece à seguinte ordem: not primeiro, and em seguida e or por último (BANIN, 2018).

Veja o exemplo a seguir:

**In [5]**

A **=** 15

B **=** 9

C **=** 9

print(B **==** C or A **<** B and A **<** C)

print((B **==** C or A **<** B) and A **<** C )

True  
  
False

Na entrada 5 (In [5]), temos os seguintes casos:

Linha 5: True or False and False. Nesse caso será feito primeiro o and, que resulta em False, mas em seguida é feito o or, que então resulta em verdadeiro.

Linha 6: (True or False) and False. Nesse caso será feito primeiro o or, que resulta em True, mas em seguida é feito o and, que então resulta em falso, pois ambos teriam que ser verdadeiro para o and ser True.

Utilize o [emulador](https://trinket.io/python3/c5256b490a" \t "_blank) para testar os códigos. Crie novas variáveis e condições para treinar.

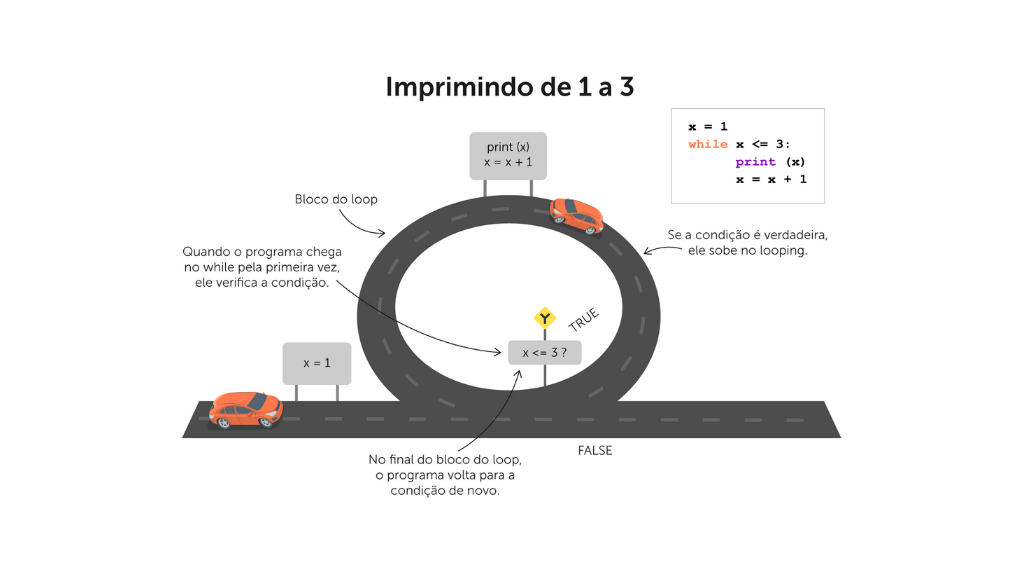
**Estruturas de repetição em Python: While e For**



É comum, durante a implementação de uma solução, criarmos estruturas que precisam executar várias vezes o mesmo trecho de código. Tal estrutura é chamada de estrutura de repetição e, em *Python*, pode ser feita pelo comando *while*(enquanto) ou pelo comando for (para).

Antes de apresentarmos a implementação, observe a figura abaixo. Temos a ilustração de uma estrutura de repetição por meio de uma montanha russa. Enquanto uma condição for verdadeira, ou seja, enquanto x for menor ou igual a 3, o condutor continuará no loop; quando for falsa, ou seja, quando x for maior que 3, então ele sairá.

Em uma estrutura de repetição sempre haverá uma estrutura decisão, pois a repetição de um trecho de código sempre está associada a uma condição. Ou seja, um bloco de comandos será executado repetidas vezes, até que uma condição não seja mais satisfeita.

Fluxograma estrutura de repetição. Fonte: adaptada de Griffiths e Barry (2009, p. 28).

O comando while deve ser utilizado para construir e controlar a estrutura decisão, sempre que o número de repetições não seja conhecido. Por exemplo, quando queremos solicitar e testar se o número digitado pelo usuário é par ou ímpar. Quando ele digitar zero, o programa se encerra. Veja, não sabemos quando o usuário irá digitar, portanto somente o while é capaz de solucionar esse problema. Veja a implementação desse problema a seguir.

**In [6]**

numero **=** 1

while numero **!=** 0:

    numero **=** int(input("Digite um número: "))

    if numero **%** 2 **==** 0:

        print("Número par!")

    else:

        print("Número ímpar!")

Digite um número: 4  
  
Número par!  
  
Digite um número: 7  
  
Número ímpar!  
  
Digite um número: 0  
  
Número par!

A seguir, [você pode testar o código](http://pythontutor.com/" \t "_blank), verificando o passo a passo de sua execução. Ao clicar em "Next", você irá para a próxima linha a ser executada do "while" para cada um dos casos. Ao clicar em "Prev", você retorna a linha executada anteriormente.

Observe que a seta verde na linha de código representa a linha em execução, e a seta vermelha representa a próxima linha a ser executada. Digite um número, submeta, clique em "Next" e observe o fluxo de execução e as saídas do código.

Veja a construção do comando while na entrada 6 (In [6]). Na linha 1, criamos uma variável chamada número com valor 1. Na linha 2 criamos a estrutura de repetição, veja que o comando while possui uma condição: o número tem que ser diferente de zero para o bloco executar. Todo o bloco com a identação de uma tabulação (4 espaços) faz parte da estrutura de repetição. Lembre: todos os blocos de comandos em Python são controlados pela indentação.

Na prática é comum utilizarmos esse tipo de estrutura de repetição, com while, para deixarmos serviços executando em servidores. Por exemplo, o serviço deve ficar executando enquanto a hora do sistema for diferente de 20 horas.

Outro comando muito utilizado para construir as estruturas de repetição é o for. A instrução for em *Python*difere um pouco do que ocorre em outras linguagens, como C ou Java, com as quais você pode estar acostumado [(veja aqui)](https://docs.python.org/3/tutorial/controlflow.html" \t "_blank).

Em vez de sempre dar ao usuário a capacidade de definir a etapa de iteração e a condição de parada (como C), a instrução *Python for*itera sobre os itens de qualquer sequência, por exemplo, iterar sobre os caracteres de uma palavra, pois uma palavra é um tipo de sequência. Observe um exemplo a seguir.

**In [7]**

nome **=** "Guido"

for c in nome:

    print(c)

G  
  
u  
  
i  
  
d  
  
o

Na entrada 7 (In [7]) construímos nossa primeira estrutura de repetição com for. Vamos analisar a sintaxe da estrutura. Usamos o comando "*for*" seguido da variável de controle "c", na sequência o comando "in", que traduzindo significa "em", por fim, a sequência sobre a qual a estrutura deve iterar. Os dois pontos marcam o início do bloco que deve ser repetido.

Junto com o comando for, podemos usar a função *enumerate*() para retornar à posição de cada item, dentro da sequência. Considerando o exemplo dado, no qual atribuímos a variável "nome" o valor de "Guido", "G" ocupa a posição 0 na sequência, "u" ocupa a posição 1, "i" a posição 2, e assim por diante. A função *enumerate*() recebe como parâmetro a sequência e retorna a posição.

Para que possamos capturar tanto a posição quanto o valor, vamos precisar usar duas variáveis de controle. Observe o código a seguir, veja que a variável "i" é usada para capturar a posição e a variável "c" cada caracter da palavra.

**In [8]**

nome **=** "Guido"

for i, c in enumerate(nome):

    print(f"Posição = {i}, valor = {c}")

Posição = 0, valor = G  
  
Posição = 1, valor = u  
  
Posição = 2, valor = i  
  
Posição = 3, valor = d  
  
Posição = 4, valor = o

**Controle de repetição com Range, Break e Continue**



Como já dissemos, o comando for em *Python*requer uma sequência para que ocorra a iteração. Mas e se precisarmos que a iteração ocorra em uma sequência numérica, por exemplo, de 0 a 5? Para criar uma sequência numérica de iteração em *Python*, podemos usar a função range(). Observe o código a seguir.

**In [9]**

for x in range(5):

    print(x)

0  
  
1  
  
2  
  
3  
  
4

No comando, "x" é a variável de controle, ou seja, a cada iteração do laço, seu valor é alterado, já a função range() foi utilizada para criar um "iterable" numérico (objeto iterável) para que as repetições acontecesse. A função range() pode ser usada de três formas distintas:

1. método 1: passando um único argumento que representa a quantidade de vezes que o laço deve repetir;
2. método 2: passando dois argumentos, um que representa o início das repetições e outro o limite superior (NÃO INCLUÍDO) do valor da variável de controle;
3. método 3: Passando três argumentos, um que representa o início das repetições; outro, o limite superior (NÃO INCLUÍDO) do valor da variável de controle e um que representa o incremento. Observe as três maneiras a seguir.

Além de controlar as iterações com o tamanho da sequência, outra forma de influenciar no fluxo é por meio dos comandos "break" e "continue". O comando break para a execução de uma estrutura de repetição, já com o comando continue, conseguimos "pular" algumas execuções, dependendo de uma condição. Veja o exemplo:

**In [10]**

*# Exemplo de uso do break*

disciplina **=** "Linguagem de programação"

for c in disciplina:

    if c **==** 'a':

        break

    else:

        print(c)

L  
  
i  
  
n  
  
g  
  
u

No exemplo com uso do *break*, perceba que foram impressas as letras L i n g u, quando chegou a primeira letra a, a estrutura de repetição foi interrompida. Agora observe o exemplo com continue.

**In [11]**

# Exemplo de uso do continue

disciplina **=** "Linguagem de programação"

for c in disciplina:

    if c **==** 'a':

        continue

    else:

        print(c)

L  
  
i  
  
n  
  
g  
  
u  
  
g  
  
e  
  
m  
  
   
  
d  
  
e  
  
   
  
p  
  
r  
  
o  
  
g  
  
r  
  
m  
  
ç  
  
ã  
  
o

No exemplo com uso do continue, perceba que foram impressas todas as letras, exceto as vogais "a", pois toda vez que elas eram encontradas, o continue determina que se pule, mas que a repetição prossiga para o próximo valor. Utilize o [emulador](https://trinket.io/python3/ee7e47d606" \t "_blank) para executar os códigos e crie novas combinações para explorar.

\_\_\_\_\_

**📝 Exemplificando**

Vamos brincar um pouco com o que aprendemos. Vamos criar uma solução que procura pelas vogais "a", "e" em um texto (somente minúsculas). Toda vez que essas vogais são encontradas, devemos informar que encontramos e qual posição do texto ela está.

Nosso texto será: texto = A inserção de comentários no código do programa é uma prática normal. Em função disso, toda linguagem de programação tem alguma maneira de permitir que comentários sejam inseridos nos programas. O objetivo é adicionar descrições em partes do código, seja para documentá-lo ou para adicionar uma descrição do algoritmo implementado (BANIN, 2018, p. 45).

**In [12]**

**texto = """**

**A inserção de comentários no código do programa é uma prática normal.**

**Em função disso, toda linguagem de programação tem alguma maneira de permitir que comentários sejam inseridos nos programas.**

**O objetivo é adicionar descrições em partes do código, seja para documentá-lo ou para adicionar uma descrição do algoritmo implementado (BANIN, p. 45, 2018)."**

**"""**

**for i, c in enumerate(texto):**

**if c == 'a' or c == 'e':**

**print(f"Vogal '{c}' encontrada, na posição {i}")**

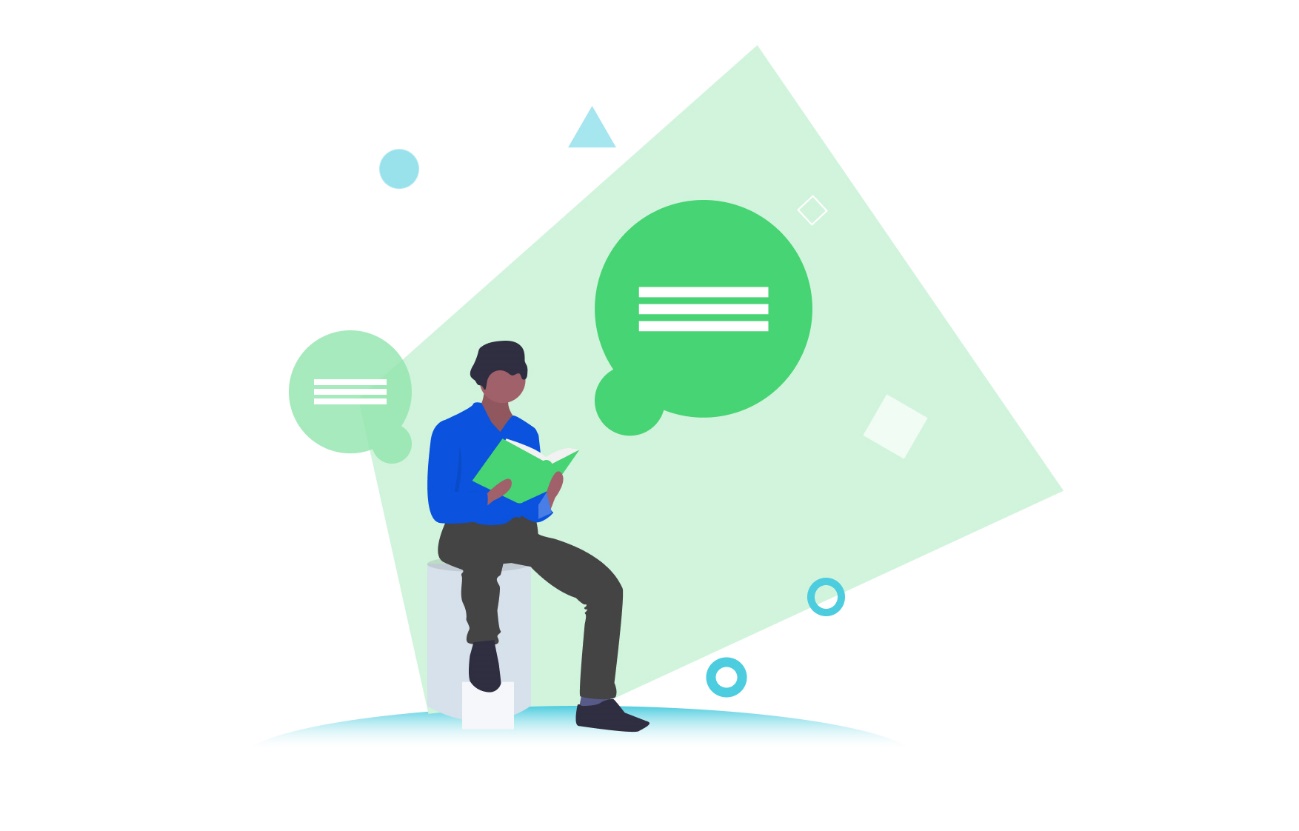
**else:**

**continue**

Vogal 'e' encontrada, na posição 6  
  
Vogal 'e' encontrada, na posição 13  
  
Vogal 'e' encontrada, na posição 18  
  
Vogal 'a' encontrada, na posição 45  
  
Vogal 'a' encontrada, na posição 47  
  
Vogal 'a' encontrada, na posição 53  
  
Vogal 'a' encontrada, na posição 61  
  
Vogal 'a' encontrada, na posição 67  
  
Vogal 'a' encontrada, na posição 91  
  
Vogal 'a' encontrada, na posição 98  
  
Vogal 'e' encontrada, na posição 100  
  
Vogal 'e' encontrada, na posição 104  
  
Vogal 'a' encontrada, na posição 111  
  
Vogal 'a' encontrada, na posição 113  
  
Vogal 'e' encontrada, na posição 119  
  
Vogal 'a' encontrada, na posição 122  
  
Vogal 'a' encontrada, na posição 127  
  
Vogal 'a' encontrada, na posição 130  
  
Vogal 'e' encontrada, na posição 132  
  
Vogal 'a' encontrada, na posição 135  
  
Vogal 'e' encontrada, na posição 138  
  
Vogal 'e' encontrada, na posição 141  
  
Vogal 'e' encontrada, na posição 151  
  
Vogal 'e' encontrada, na posição 156  
  
Vogal 'e' encontrada, na posição 166  
  
Vogal 'a' encontrada, na posição 168  
  
Vogal 'e' encontrada, na posição 174  
  
Vogal 'a' encontrada, na posição 190  
  
Vogal 'a' encontrada, na posição 192  
  
Vogal 'e' encontrada, na posição 201  
  
Vogal 'a' encontrada, na posição 209  
  
Vogal 'a' encontrada, na posição 216  
  
Vogal 'e' encontrada, na posição 220  
  
Vogal 'e' encontrada, na posição 227  
  
Vogal 'e' encontrada, na posição 230  
  
Vogal 'a' encontrada, na posição 234  
  
Vogal 'e' encontrada, na posição 237  
  
Vogal 'e' encontrada, na posição 252  
  
Vogal 'a' encontrada, na posição 254  
  
Vogal 'a' encontrada, na posição 257  
  
Vogal 'a' encontrada, na posição 259  
  
Vogal 'e' encontrada, na posição 266  
  
Vogal 'a' encontrada, na posição 278  
  
Vogal 'a' encontrada, na posição 280  
  
Vogal 'a' encontrada, na posição 282  
  
Vogal 'a' encontrada, na posição 289  
  
Vogal 'a' encontrada, na posição 294  
  
Vogal 'e' encontrada, na posição 297  
  
Vogal 'a' encontrada, na posição 309  
  
Vogal 'e' encontrada, na posição 323  
  
Vogal 'e' encontrada, na posição 325  
  
Vogal 'a' encontrada, na posição 328

Uma novidade nesse exemplo foi o uso das aspas triplas para que pudéssemos quebrar a linha do texto. As aspas triplas também podem ser usadas para criar comentários de várias linhas em *Python*. [Que tal alterar a solução de busca por vogais e contar quantas vezes cada vogal foi encontrada](https://trinket.io/python3/402eca8dbc). Lembre-se de que, quanto mais praticar, mais desenvolverá sua lógica de programação e fluência em uma determinada linguagem!

**Conclusão**



Em seu novo projeto, você foi designado a implementar uma solução que envolve o cálculo do imposto de renda, baseado nos dados do ano de 2020. O primeiro passo é encontrar uma fonte confiável de informação, que nesse caso é o portal da [Receita Federal](http://receita.economia.gov.br/acesso-rapido/tributos/irpf-imposto-de-renda-pessoa-fisica). Você encontra vários links a respeito desse imposto, e [clicando aqui](http://receita.economia.gov.br/acesso-rapido/tributos/irpf-imposto-de-renda-pessoa-fisica#calculo_mensal_IRPF)você tem acesso às tabelas de incidência mensal desse imposto para cada ano. Como o ano solicitado foi 2020, a última tabela disponível é a de 2015:

Tabela de incidência mensal, ano 2015. Fonte: Receita Federal.

Com a tabela de incidência mensal em mãos, é hora de escolher a ferramenta que irá implementar a solução, pensar no algoritmo e fazer a implementação. Você pode usar o *Colab*, uma vez que não precisa instalar e por ser online, pode compartilhar o resultado, tanto com seu gestor quanto com o cliente.

O programa deve receber um salário, com base no valor informado, e o algoritmo deve verificar em qual faixa do imposto esse valor se enquadra, quando encontrar a faixa, o programa imprime o valor a ser deduzido. Pois bem, agora vamos implementar esse algoritmo, observe o código a seguir.

**In [1]**

salario **=** 0

salario **=** float(input("Digite o salário do colaborador: "))

if salario **<=** 1903.98:

    print(f"O colaborador isento de imposto.")

elif salario **<=** 2826.65:

    print(f"O colaborador deve pagar R$ 142,80 de imposto.")

elif salario **<=** 3751.05:

    print(f"O colaborador deve pagar R$ 354,80 de imposto.")

elif salario **<=** 4664.68:

    print(f"O colaborador deve pagar R$ 636,13 de imposto.")

else:

    print(f"O colaborador deve pagar R$ 869,36 de imposto.")

Digite o salário do colaborador: 2000  
  
O colaborador deve pagar R$ 142,80 de imposto.

Na solução apresentada, veja que na linha 1, criamos e inicializamos a variável "salario", embora não seja obrigatório em *Python*é uma boa prática de programação.

Veja como fizemos o teste do valor, por que usamos somente um valor para testar? Fizemos dessa forma, porque em uma estrutura condicional encadeada, quando a primeira condição for satisfeita, as demais não são testadas. Por exemplo, considerando um colaborada que ganha R$ 2 mil por mês, embora esse valor seja menor que 3751,05 e 4664,68, a primeira condição que é satisfeita para o caso é a da linha 6 "elif salario <= 2826.65", logo essa será executada e não as demais.

Que tal mostrar para seu gerente e cliente sua pró-atividade e, além de criar uma solução que exibe o valor do desconto, também já seja impresso o valor do salário com o desconto aplicado? Observe o código com essa nova *feature*!

**In [2]**

**salario = 0**

**salario = float(input("Digite o salário do colaborador: "))**

**if salario <= 1903.98:**

**print(f"O colaborador isento de imposto.")**

**print(f"Salário a receber = {salario}")**

**elif salario <= 2826.65:**

**print(f"O colaborador deve pagar R$ 142,80 de imposto.")**

**print(f"Salário a receber = {salario - 142.80}")**

**elif salario <= 3751.05:**

**print(f"O colaborador deve pagar R$ 354,80 de imposto.")**

**print(f"Salário a receber = {salario - 354.80}")**

**elif salario <= 4664.68:**

**print(f"O colaborador deve pagar R$ 636,13 de imposto.")**

**print(f"Salário a receber = {salario - 636.13}")**

**else:**

**print(f"O colaborador deve pagar R$ 869,36 de imposto.")**

**print(f"Salário a receber = {salario - 869.36}")**

Digite o salário do colaborador: 2000  
  
O colaborador deve pagar R$ 142,80 de imposto.  
  
Salário a receber = 1857.2

\_\_\_\_\_

**➕ Pesquise mais**

Que tal treinar um pouco mais de programação e conhecer novas fontes de informações? Você, estudante, tem acesso à Biblioteca Virtual, um repositório de livros e artigos que pode ser acessado.

Na página 75 (capítulo 3) da obra: Python 3 - conceitos e aplicações: uma abordagem didática, você encontra o exercício 3 (uso de condições mistas). Utilize o [emulador para](https://trinket.io/python3/402eca8dbc) resolver o desafio.