



PYTHON – AULA 1

LÓGICA DE PROGRAMAÇÃO APLICADA





emerson.abraham@fiap.com.br

Emerson R. Abraham

Professor

Doutor e Mestre em Engenharia de Produção; Especialização em Tecnologia da Informação; Bacharel em Administração de Empresas; e Certificado Profissional Scrum Master (PSM)

Professor e Pesquisador em cursos superiores como ADS, Sistemas de Informação, Automação Industrial e Gestão de Tecnologia da Informação; como pesquisador tenho trabalhado com estudos de modelagem e simulação relacionados a Lógica Fuzzy, Redes Neurais Artificiais e Algoritmos Genéticos.

AGENDA

Aula 1

Instalando e conhecendo o Python e as ferramentas do curso.
Revisitando lógica de programação em Python.

Aula 2

Revisitando lógica de programação em Python –
continuação.

Aula 3

Trabalhando com listas, tuplas e dicionários.
Leitura e escrita de JSON.

Aula 4

Leitura e escrita de arquivos.
Introdução a Numpy e Pandas.

AGENDA

Aula 5

Orientação a objeto no Python.

Aula 6

Trabalhando com os algoritmos.
Ordenação e Recursão. – Pesquisa em largura.

Aula 7

Algoritmo de Dijkstra.

Aula 8

Algoritmos Genéticos.



PREPARANDO O AMBIENTE



INSTALANDO E CONFIGURANDO
PYTHON E PYCHARM



- A primeira etapa do nosso curso consiste em instalarmos os recursos corretos e configurarmos tudo o que for necessário.
- É importante conhecer o processo de instalação e não as versões específicas que serão apresentadas aqui.
- Cada um dos recursos sofrerá atualizações por parte dos fabricantes com o passar do tempo.

- Para escrevermos programas na linguagem Python, o primeiro recurso que precisamos é o **interpretador Python**.
- Ele é um *kit*, que é composto por compiladores e bibliotecas utilizados para que o código escrito em linguagem Python possa ser reconhecido e executado com um programa.

BAIXANDO O INTERPRETADOR



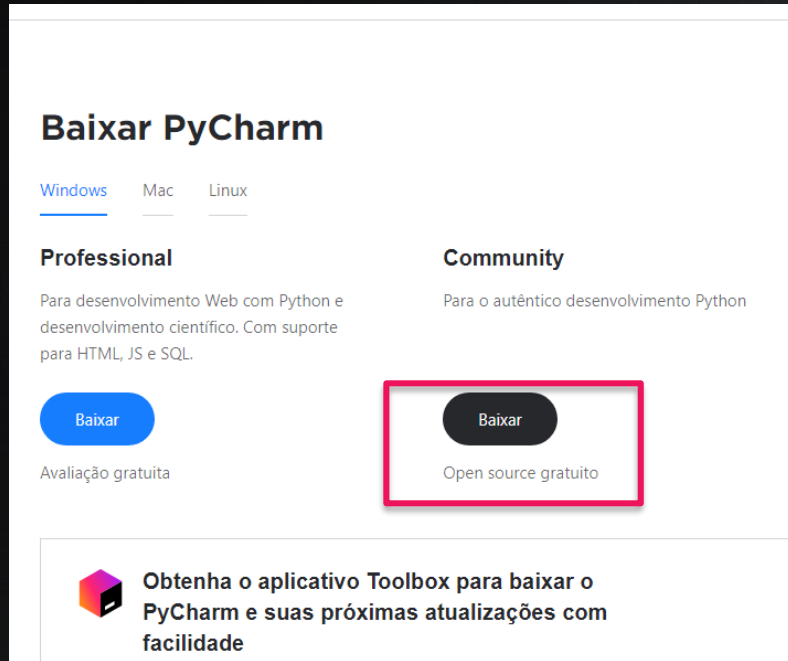
- Acesse <https://www.python.org/downloads/>
- O Python 3.x e 2.x apresentam inúmeras diferenças na forma de programar, sendo que a versão 2.x já foi descontinuada.
- Após baixar o interpretador, logo na primeira tela somos perguntados se queremos adicionar o Python ao **path**; marcar essa opção nos poupará muitas dores de cabeça, então lembre-se de fazer isso.

- Para verificar se a instalação foi feita corretamente, abra um terminal no seu sistema operacional (no caso do Windows pode ser utilizado tanto o *cmd* quanto o *PowerShell*).
- Se estiver no Windows, digite *python --version* e tecle Enter
- Se estiver no Linux, digite *python3 --version*

Agora que o nosso computador é capaz de interpretar comandos escritos em linguagem Python, podemos partir para a nossa próxima ferramenta: a **IDE**!

- A IDE (Integrated Development Environment) nada mais é do que o software onde o programador escreve os códigos que está desenvolvendo. Diferentes IDEs possuem diferentes recursos para ajudar os desenvolvedores e, naturalmente, você vai acabar se afeiçoando mais a uma específica.
- No nosso curso utilizaremos o **PyCharm**.

BAIXANDO O PYCHARM



Baixar PyCharm

Windows Mac Linux

Professional

Para desenvolvimento Web com Python e desenvolvimento científico. Com suporte para HTML, JS e SQL.

Baixar


Avaliação gratuita

Community

Para o autêntico desenvolvimento Python

Baixar

Open source gratuito

 Obtenha o aplicativo Toolbox para baixar o PyCharm e suas próximas atualizações com facilidade

- Ao acessar o link <https://www.jetbrains.com/pt-br/pycharm/download> o site oficial do JetBrains deve detectar automaticamente qual é a versão mais adequada da IDE para o seu sistema operacional.
- Selecione a opção *Community*, que é gratuita e possui todos os recursos que precisaremos ao longo das nossas aulas.
- Após baixar o instalador, realize a instalação utilizando apenas o botão de *next*.

O QUE É O PYTHON?



- Atualmente, o Python é uma das linguagens de programação mais populares do mundo (<https://www.tiobe.com/tiobe-index/>) e podemos destacar algumas de suas características que a tornam tão popular:
 - Simples e versátil; fácil de aprender
 - Suporta múltiplos paradigmas de programação
 - Tem um excelente desempenho durante a execução dos scripts
 - Comunidade e biblioteca riquíssimos.



PROGRAMANDO EM PYTHON



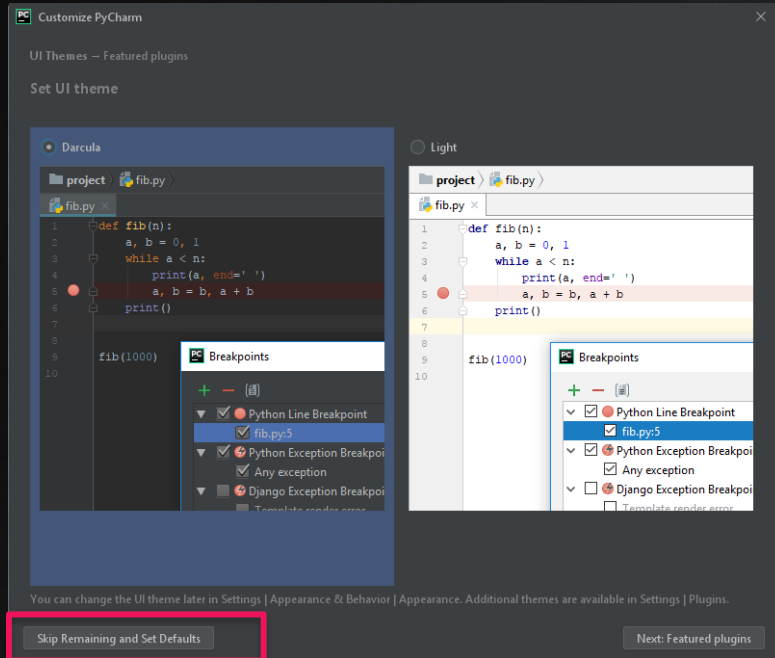
EXIBINDO
MENSAGENS NA TELA



HORA DO PRIMEIRO PROGRAMA

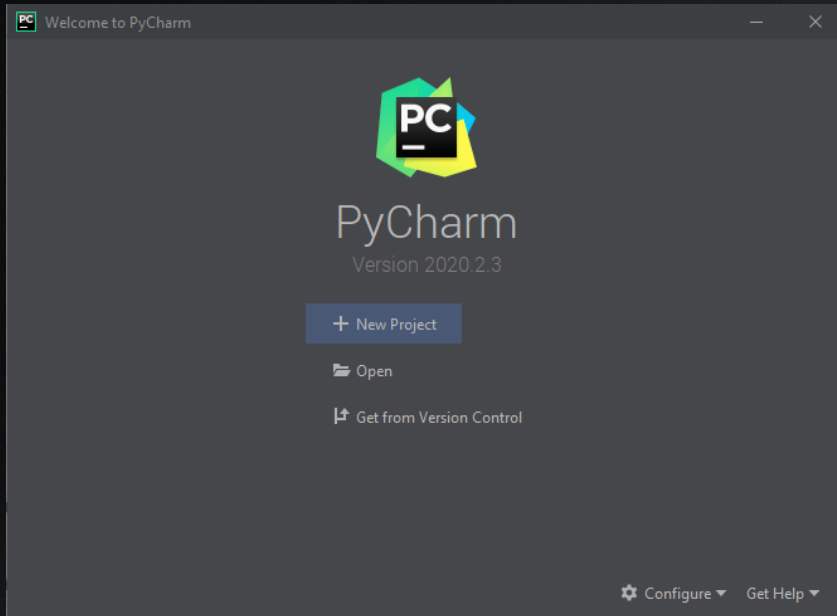
- Existe uma brincadeira entre os desenvolvedores que diz que o primeiro programa em qualquer linguagem de programação deve ser o “Hello World”.
- Para escrevermos o nosso primeiro programa, devemos abrir o PyCharm e criar um novo projeto.
- Dentro do novo projeto, criaremos um novo script.

ABRINDO O PYCHARM



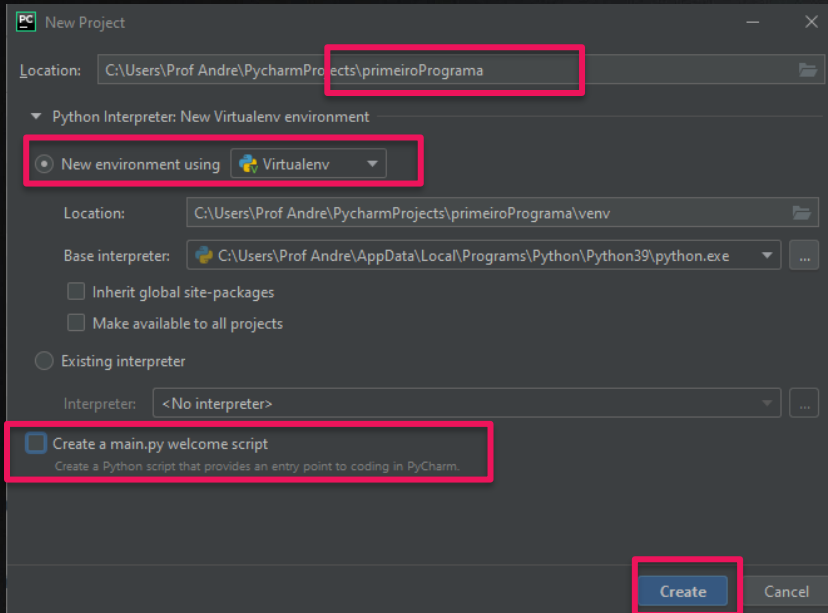
- Ao abrirmos o PyCharm pela primeira vez, poderemos fazer uma série de configurações.
- Para os recursos que utilizaremos ao longo das nossas aulas, podemos selecionar a opção de iniciar com as configurações padrão.

CRIANDO UM PROJETO NO PYCHARM



- Na primeira execução do PyCharm poderemos selecionar a opção *New Project* logo na tela inicial.
- Essa mesma opção poderá ser acessada, posteriormente, através do menu *File* → *New Project*.

CRIANDO UM PROJETO NO PYCHARM

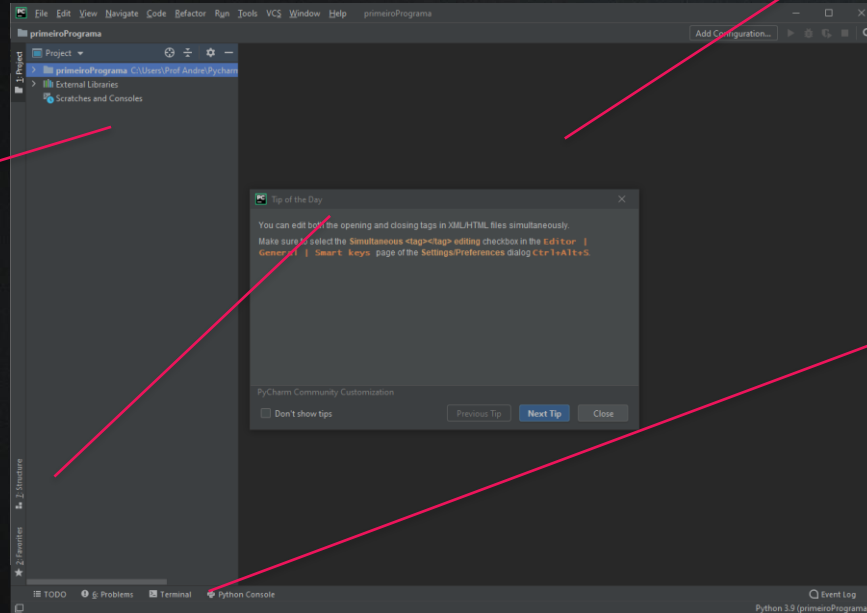


- Na caixa identificada como *Location* podemos verificar o local em que o novo projeto será criado no nosso computador.
- O nome padrão é *pythonProject*, mas vamos alterar para *primeiroPrograma*
- Além disso o PyCharm cria para nós um novo ambiente virtual. Futuramente entenderemos melhor as vantagens e desvantagens desse recurso, mas por hoje deixaremos ativado.
- Também desmarcaremos a caixa *Create a main.py welcomeScript*, para que nosso projeto seja criado sem nenhum código já escrito.
- Ao final, clique em *Create*

CONHECENDO O PYCHARM

A aba *Project* é onde poderemos visualizar todos os arquivos que fazem parte do nosso projeto.

Por padrão o PyCharm exibe dicas diárias ao ser aberto. Você pode optar por marcar a caixa “Don’t show tips” para não receber mais as dicas

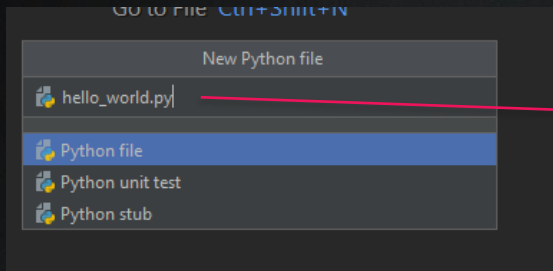
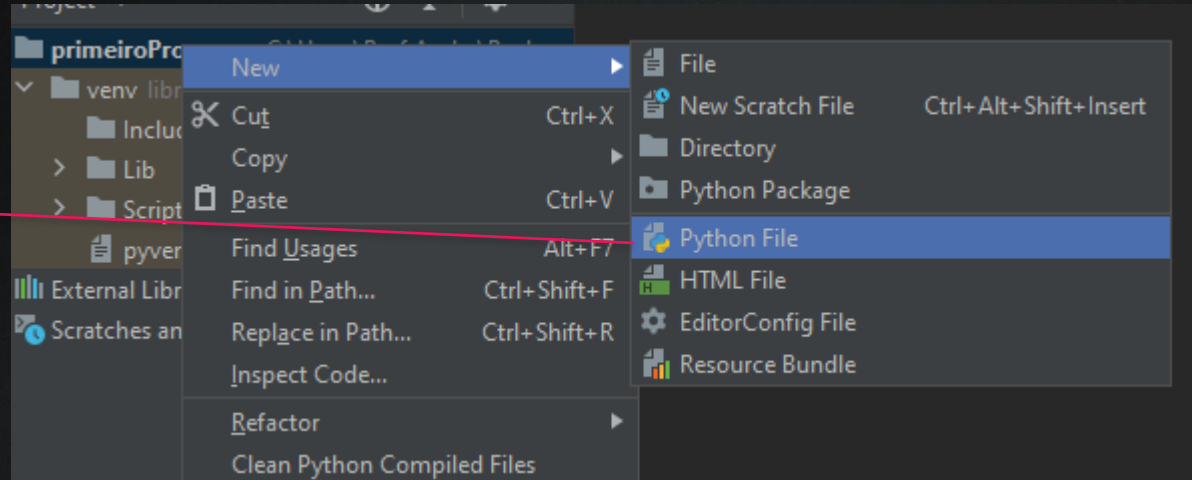


Quando criarmos nosso primeiro script, seu conteúdo será exibido aqui

Quando executarmos um script, veremos os resultados aqui no terminal

CRIANDO UM NOVO SCRIPT

Você pode iniciar um Script ao clicar sobre o nome do seu projeto com o botão direito do mouse e selecionar a opção New → Python File

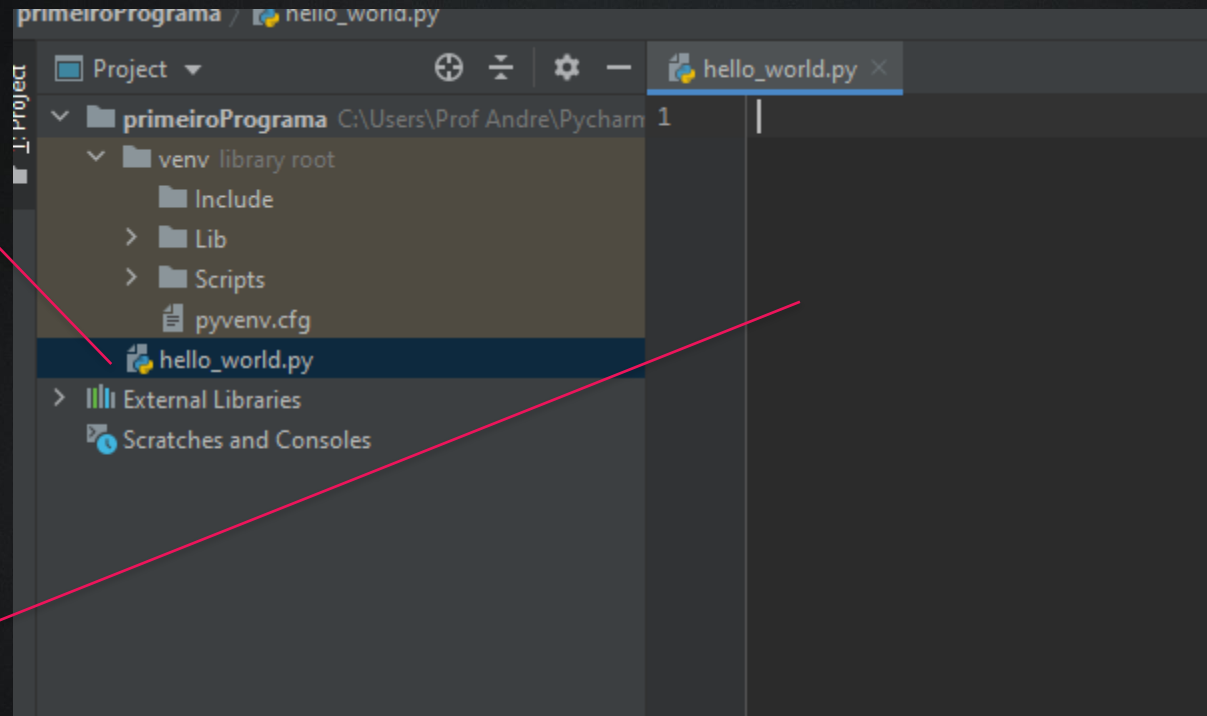


Na janela que for exibida, devemos dar um nome para nosso novo Script. Chamaremos de *hello_world.py*

CRIANDO UM NOVO SCRIPT

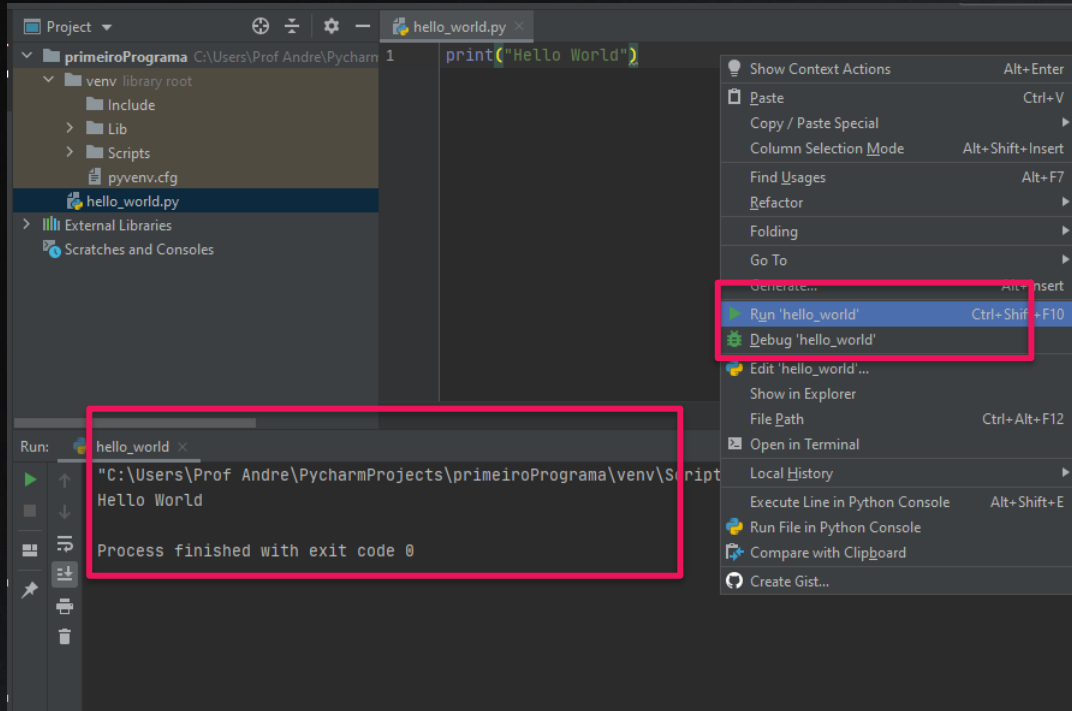
Note que imediatamente nosso novo arquivo apareceu dentro da pasta do projeto

E o Script foi aberto para podermos escrever códigos dentro dele!



```
print("Hello World")
```

HORA DE RODAR!



- Para rodar nosso programa, podemos teclar **ALT + SHIFT + F10**, clicar com o botão direito sobre o script e clicar em **Run** ou ainda usar o menu **Run** e a opção **Run**.
- Ao fazermos isso, o script executado será mostrado na parte inferior da tela.



PROGRAMANDO EM PYTHON

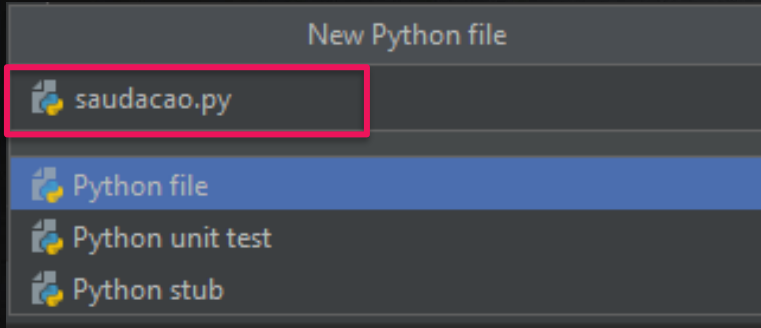


INTERAGINDO COM O
USUÁRIO



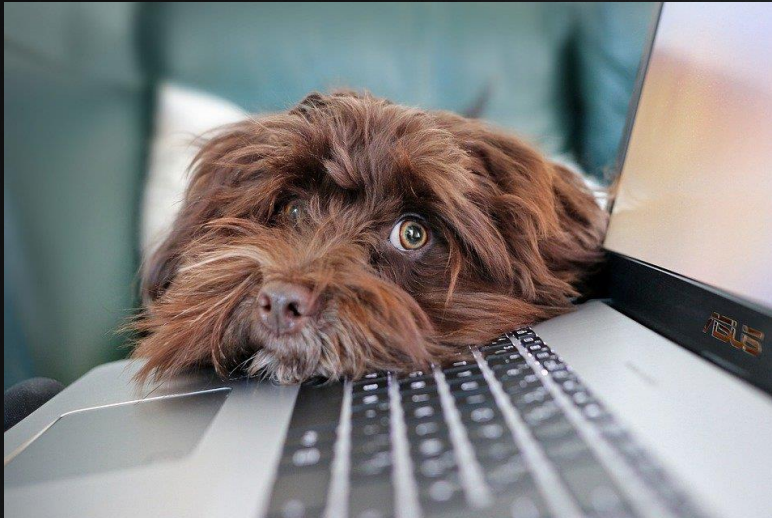
UM POUCO DE INTERAÇÃO

- Toda vez que o usuário rodar esse programa, a mesma mensagem será exibida. E se houver uma forma de nós personalizarmos a saudação à partir do nome do usuário?
- Para isso vamos criar um novo script dentro do nosso projeto
- Nosso script vai receber o nome do usuário através do teclado
- Vamos armazenar o nome do usuário em uma variável
- Por fim, vamos exibir o nome do usuário na tela



- Para nosso próximo exercício vamos criar um script chamado **saudacao.py**
- Note que o nome do script, além de obedecer à regra de iniciar com letra minúscula, não contém acentos e nem caracteres latinos.
- É uma boa prática de programação evitar qualquer tipo de caractere especial nos nomes das classes, arquivos e pastas.

LENDO ATRAVÉS DO **TECLADO**

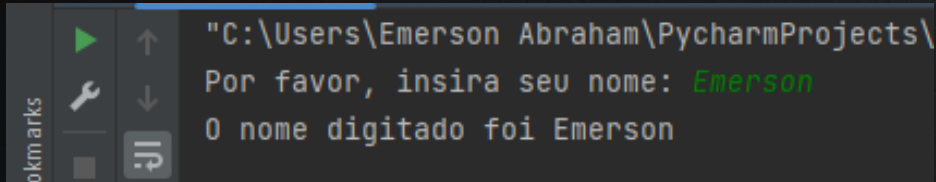


- Para lermos dados através do teclado, vamos utilizar uma função chamada *input()*
- Com esse recurso vamos ser capazes de capturar as informações que o usuário digitar até o momento em que ele teclar “Enter”.
- Esse será nosso principal meio de interação com o usuário ao longo de todo esse curso



```
input("Por favor, insira seu nome: ")
```


EXECUTANDO O PROGRAMA



```
"C:\Users\Emerson Abraham\PycharmProjects\  
Por favor, insira seu nome: Emerson  
O nome digitado foi Emerson
```

- Ao executarmos nosso script, notamos que é possível interagir na parte inferior do PyCharm.
- Como a linha *input* do nosso código permite que o usuário digite algo pelo seu teclado, nosso programa fica aguardando o usuário digitar no console e teclar *Enter*.
- Note, porém, que após o usuário digitar seu nome e teclar *Enter*, nada mais acontece e o programa é terminado.
- Para exibirmos na tela o nome do usuário, vamos precisar criar uma **variável**.

- Uma **variável** é um espaço criado na memória RAM do seu computador onde o programa será capaz de armazenar dados enquanto está em execução.
- Em linguagem Python não precisamos criar as variáveis indicando seus tipos.
- Elas são criadas ao escrever seus nomes e atribuir um valor.



```
nome = input("Por favor, insira seu nome: ")  
print("O nome digitado foi " + nome)
```

Ao escrevermos *nome* = o Python sabe que deve criar uma variável chamada nome e que ela deve ter o tipo correspondente ao valor que vier após o sinal de igual.



PROGRAMANDO EM PYTHON



TRABALHANDO COM
NÚMEROS



TRABALHANDO COM NÚMEROS

- Até agora nossos scripts trabalharam apenas com texto. Mas o que ocorre se quisermos manipular valores numéricos?
- Por essa razão, vamos criar um programa “calculadora”, que receba dois valores numéricos e realize as 4 operações matemáticas entre eles.

Assim como fizemos com a os dois
exercícios anteriores, crie o script
calculadora.py.

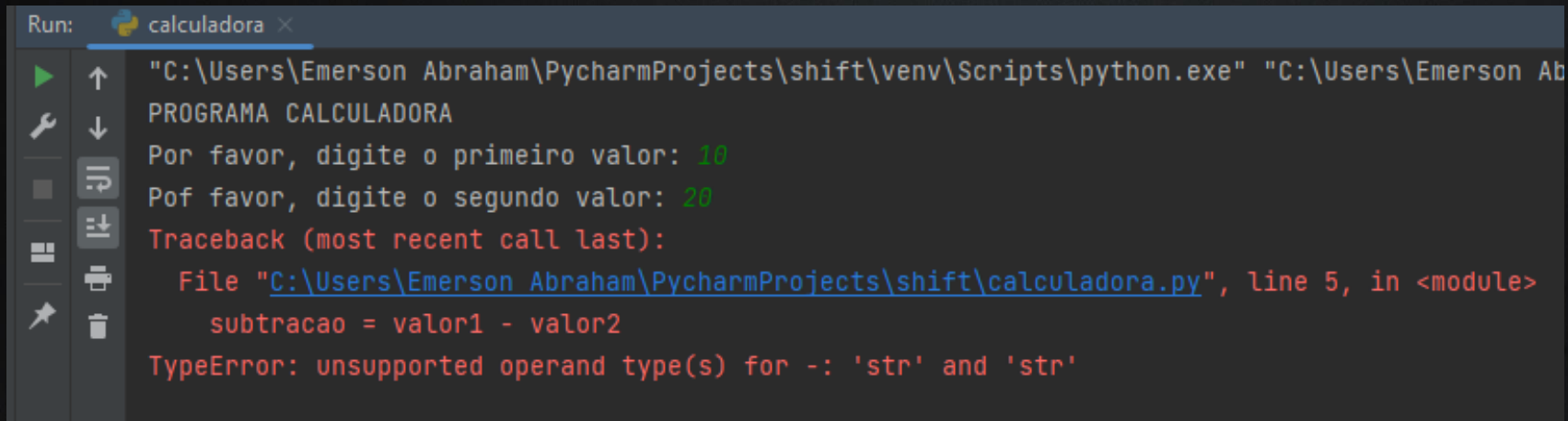


```
print("PROGRAMA CALCULADORA")
valor1 = input("Por favor, digite o primeiro valor: ")
valor2 = input("Pof favor, digite o segundo valor: ")
soma = valor1 + valor2
subtracao = valor1 - valor2
divisao = valor1 / valor2
multiplicacao = valor1 * valor2

print("A soma entre os dois valores é de " + soma)
print("A subtração entre os dois valores é de " + subtracao)
print("A divisão entre os dois valores é de " + divisao)
print("A multiplicação entre os dois valores é de " + multiplicacao)
```

Nosso programa se inicia exibindo uma mensagem ao usuário

Ao executar o nosso programa, teremos um péssimo resultado:



The screenshot shows a terminal window titled "Run: calculadora". The output of the program is as follows:

```
"C:\Users\Emerson Abraham\PycharmProjects\shift\venv\Scripts\python.exe" "C:\Users\Emerson Ab
PROGRAMA CALCULADORA
Por favor, digite o primeiro valor: 10
Pof favor, digite o segundo valor: 20
Traceback (most recent call last):
  File "C:\Users\Emerson Abraham\PycharmProjects\shift\calculadora.py", line 5, in <module>
    subtracao = valor1 - valor2
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

The error message is displayed in red text, indicating a `TypeError: unsupported operand type(s) for -: 'str' and 'str'`. The program prompts the user for two values, 10 and 20, which are entered as strings.

O erro em vermelho nos diz que não é possível utilizar o operador – para variáveis do tipo `str`(String)



```
print("PROGRAMA CALCULADORA")
valor1 = input("Por favor, digite o primeiro valor")
valor2 = input("Pof favor, digite o segundo valor")
#soma = valor1 + valor2
#subtracao = valor1 - valor2
#divisao = valor1 / valor2
#multiplicacao = valor1 * valor2

#print("A soma entre os dois valores é de " + soma)
#print("A subtração entre os dois valores é de " + subtracao)
#print("A divisão entre os dois valores é de " + divisao)
#print("A multiplicação entre os dois valores é de " + multiplicacao)
```

Acrescente um símbolo de # na frente de todas as linhas de cálculo e de exibição de mensagem.

Esse é o símbolo de comentário, e faz com que a linha seja ignorada.

```
print("PROGRAMA CALCULADORA")
valor1 = input("Por favor, digite o primeiro valor")
valor2 = input("Por favor, digite o segundo valor")
```

```
print(type(valor1))
```

```
#soma = valor1 + valor2
#subtracao = valor1 - valor2
#divisao = valor1 / valor2
#multiplicacao = valor1 * valor2
```

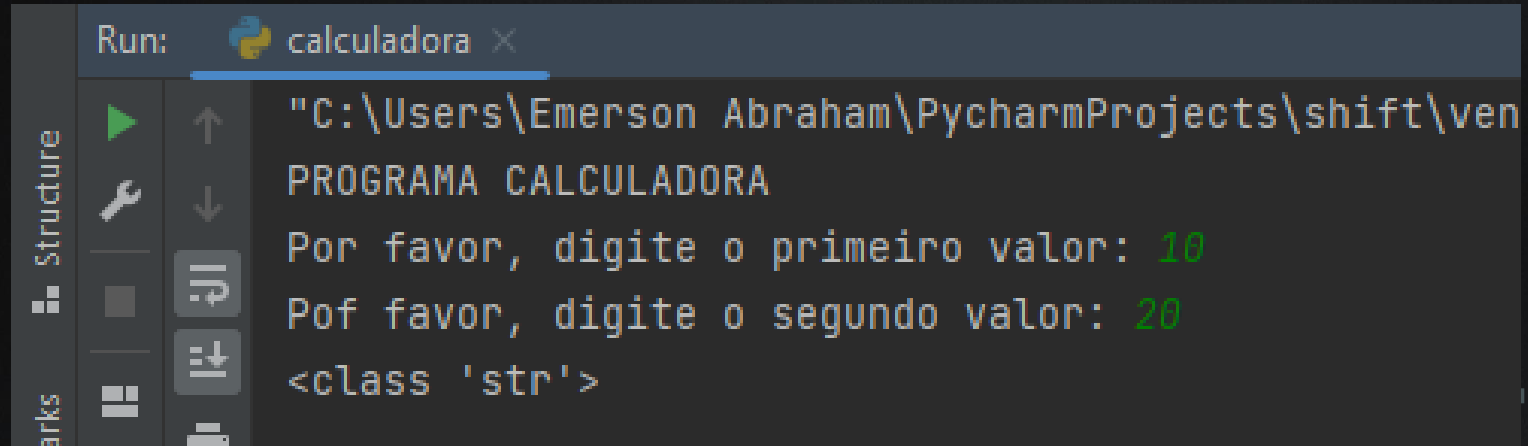
```
#print("A soma entre os dois valores é de " + soma)
#print("A subtração entre os dois valores é de " + subtracao)
#print("A divisão entre os dois valores é de " + divisao)
#print("A multiplicação entre os dois valores é de " + multiplicacao)
```

Agora vamos *printar* na tela o tipo da variável *valor1*.

Faremos isso utilizando a função *type()* e escrevendo entre parênteses a variável da qual queremos saber o tipo.



Ao executarmos nosso programa, podemos verificar que a variável *valor1* é de fato do tipo *str*.



The screenshot shows a PyCharm Run console window titled "Run: calculadora". The console output is as follows:

```
"C:\Users\Emerson Abraham\PycharmProjects\shift\ven  
PROGRAMA CALCULADORA  
Por favor, digite o primeiro valor: 10  
Pof favor, digite o segundo valor: 20  
<class 'str'>
```

The left sidebar of the IDE shows the "Structure" and "arks" (likely "Tools") panels. The "Structure" panel contains icons for running (green play button), debugging (wrench), and other development tools. The "arks" panel contains icons for file explorer, search, and other IDE functions.

A função *input()* faz com que qualquer dado digitado pelo usuário seja tratado como um **texto**.

Se quisermos manipular esse dado como número, precisaremos fazer uma **conversão**.

Vamos conhecer alguns tipos de dados que podem nos ajudar?

Para números inteiros podemos utilizar o tipo *int*.
Já para números com casas decimais, podemos utilizar o tipo *float*.

A conversão pode ser realizada escrevendo *float(valor a ser convertido)* ou *int(valor a ser convertido)*.

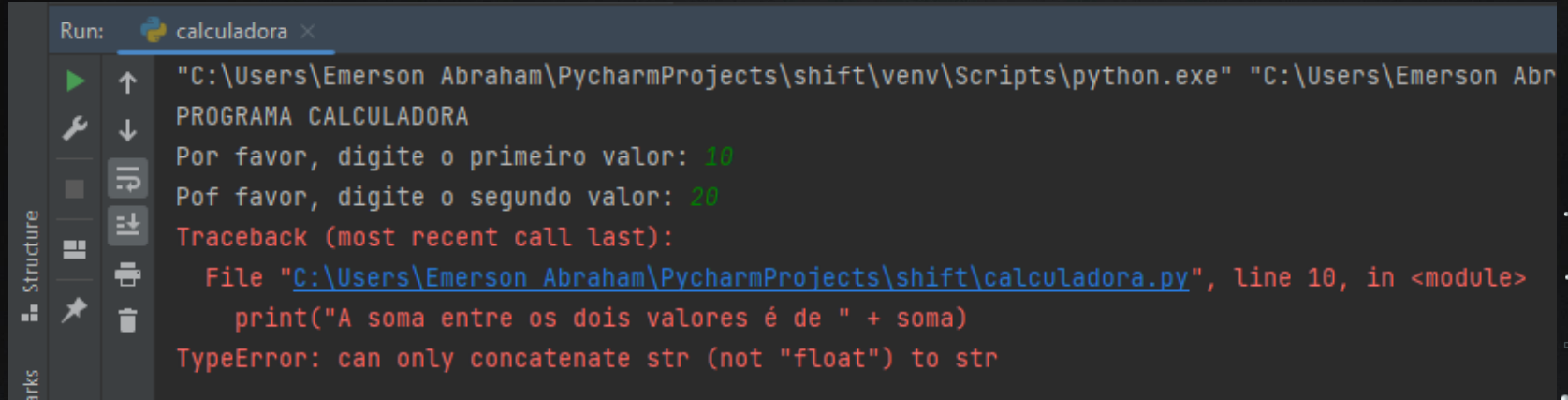


```
print("PROGRAMA CALCULADORA")
valor1 = float(input("Por favor, digite o primeiro valor"))
valor2 = float(input("Pof favor, digite o segundo valor"))

soma = valor1 + valor2
subtracao = valor1 - valor2
divisao = valor1 / valor2
multiplicacao = valor1 * valor2

print("A soma entre os dois valores é de " + soma)
print("A subtração entre os dois valores é de " + subtracao)
print("A divisão entre os dois valores é de " + divisao)
print("A multiplicação entre os dois valores é de " + multiplicacao)
```

O novo erro nos indica que o *print* não pode ser realizado, pois a variável *soma* agora é do tipo *float* e só podemos utilizar o *+* para printar variáveis do tipo String.



The screenshot shows the Run console of a PyCharm IDE. The title bar indicates the file being executed is 'calculadora.py'. The output shows the program's execution flow: it prints 'PROGRAMA CALCULADORA', prompts for two values (10 and 20), and then encounters a 'TypeError: can only concatenate str (not "float") to str'. The error message is displayed in red text, indicating a runtime exception. The traceback points to line 10 of the file, where an attempt is made to concatenate a string with a float variable named 'soma'.

```
Run: calculadora x
"C:\Users\Emerson Abraham\PycharmProjects\shift\venv\Scripts\python.exe" "C:\Users\Emerson Abr
PROGRAMA CALCULADORA
Por favor, digite o primeiro valor: 10
Pof favor, digite o segundo valor: 20
Traceback (most recent call last):
  File "C:\Users\Emerson Abraham\PycharmProjects\shift\calculadora.py", line 10, in <module>
    print("A soma entre os dois valores é de " + soma)
TypeError: can only concatenate str (not "float") to str
```

A solução para o nosso problema é utilizarmos o método *format* para exibir um texto.

A ideia é simples: nos lugares onde quisermos exibir uma variável ao longo de um texto, devemos colocar um sinal de abrir e fechar chaves (*{}*), como um *marcador*.



Depois, utilizamos a função *format* para *preencher* esse marcador com o valor que desejamos exibir.




```
print("PROGRAMA CALCULADORA")
valor1 = float(input("Por favor, digite o primeiro valor"))
valor2 = float(input("Pof favor, digite o segundo valor"))

soma = valor1 + valor2
subtracao = valor1 - valor2
divisao = valor1 / valor2
multiplicacao = valor1 * valor2

print("A soma entre os dois valores é de {}".format(soma))
print("A subtração entre os dois valores é de {}".format(subtracao))
print("A divisão entre os dois valores é de {}".format(divisao))
print("A multiplicação entre os dois valores é de {}".format(multiplicacao))
```



DESVIOS CONDICIONAIS: SIMPLES, COMPOSTO E ENCADEADO



- Até agora os nossos programas são todos *lineares*, ou seja, seguem um único fluxo do início ao fim.

Porém, no mundo da programação, a maior parte dos problemas são baseados em *condições*.

Imagine o seguinte cenário:

- Uma ONG criou um jogo para alertar as pessoas sobre os riscos das drogas.

Por se tratar de um assunto sensível, ficou decidido que a idade mínima para jogar é de 12 anos.

Sua função é criar um programa que receba a idade do usuário e exiba a mensagem "Você pode jogar" **caso ele tenha 12 anos ou mais.**

Para resolver esse problema, vamos criar no PyCharm (menu **File** → **New Project**) um projeto chamado **revistando_logica** e, dentro dele, um Script chamado **if_simples.py** (botão direito no nome do projeto, **New** → **Python File**).



A parte inicial do nosso código só precisa de comandos que você já aprendeu na 1ª aula: um input para permitir que o usuário digite a idade, uma variável do tipo inteiro para armazenar essa idade e alguns prints para instruir o usuário.

```
print("PROGRAMA VALIDADOR DE IDADE!")  
idade = int(input("Por favor, digite sua idade"))
```

Como faremos, porém, para usar a variável idade para verificar se o jogador pode ou não jogar?

É aí que entra o **Desvio Condicional** ou o **If**.

Os desvios condicionais são estruturas de programação que nos permitem realizar instruções dependendo do resultado de uma condição booleana.

Uma condição booleana é aquela onde o resultado só pode ser **verdadeiro** ou **falso**.



```
| if (condição booleana):  
+     instruções para serem executadas caso a condição seja verdadeira
```



```
| print("PROGRAMA VALIDADOR DE IDADE!")  
+ idade = int(input("Por favor, digite sua idade"))  
  if (idade >= 12):  
    print("Você pode jogar!")
```

- Estamos falando de desvios condicionais, no plural, mas estudamos apenas um deles.

Existem situações em que vamos querer a execução de instruções caso um teste tenha resultado **verdadeiro** e a execução de outras instruções caso um teste tenha resultado **falso**.

Para esses casos usaremos um **If Composto**. Vamos dar uma olhada em outro caso:

O estatuto de uma ONG determina que todas as doações recebidas devem gerar um valor para investimento, para cobrir momentos de necessidade.

O valor do investimento **deve ser de 5% da doação**. Porém, em **casos em que as doações ultrapassem R\$1000,00** o investimento deve ser de **15% da doação**.

Sua missão é criar um programa capaz de fazer os cálculos necessários e indicar quanto deve ser investido.

Crie um Script chamado `if_composto.py` para resolver esse problema.



Novamente a parte inicial do código só precisa de um input para permitir que o usuário digite a doação, uma variável do tipo real para armazenar essa doação e outras duas para armazenar o investimento e o dinheiro de uso imediato e alguns prints para instruir o usuário.

```
print("PROGRAMA DE GESTÃO DE DOAÇÕES")
doacao = float(input("Por favor, digite o valor da doação recebida!"));
```

```
print("A doação de R$ {} implica em um investimento de R$ {}, restando R$ {} para uso imediato".format(doacao, investimento, uso_imediato))
```

- • • • • +
- • • Como fazer a verificação dessa doação?



- + •
- + • Uma ideia seria utilizarmos dois ifs simples

```
if(doacao < 1000.00):  
    investimento = doacao * 0.05  
    usoImediato = doacao - investimento
```

```
if (doacao >= 1000.00):  
    investimento = doacao * 0.15  
    usoImediato = doacao - investimento
```

Porém, isso implica em perda de desempenho, uma vez que mesmo que o primeiro teste tenha resultado verdadeiro, o computador ainda assim verificará o segundo teste, mesmo que ele obviamente seja falso.



O If Composto é um desvio condicional que executa instruções para o caso de a condição booleana ter um resultado verdadeiro e outras instruções para o caso de a condição booleana ter um resultado falso.

```
if (condição booleana):  
    instruções para serem executadas caso a condição seja  
verdadeira  
  
else:  
    instruções para serem executadas caso a condição seja falsa
```



```
print("PROGRAMA DE GESTÃO DE DOAÇÕES")
doacao = float(input("Por favor, digite o valor da doação recebida!"));

#o if composto abaixo verifica as faixas de doação

if (doacao < 1000.00):
    investimento = doacao * 0.05
    uso_imediato = doacao - investimento
else:
    investimento = doacao * 0.15
    uso_imediato = doacao - investimento

print("A doação de R$ {} implica em um investimento de R$ {}, restando R$ {} para uso imediato".format(doacao, investimento, uso_imediato))
```

O if composto verifica se o teste *doacao < 1000.00* é verdadeiro. Caso for verdadeiro, o cálculo de 5% é feito.
Caso for falso, o cálculo de 15% é feito.

Se quisermos exibir os valores com 2 casas decimais, para que pareçam realmente moedas, podemos utilizar um módulo chamado *locale*.

Para resolvermos esse problema de forma simplificada, podemos utilizar a formatação *:.2f*, que indica que os números antes da vírgula devem ser mantidos, mas os números após a vírgula devem ser limitados a 2.



```
print("PROGRAMA DE GESTÃO DE DOAÇÕES")
doacao = float(input("Por favor, digite o valor da doação recebida!"));

#o if composto abaixo verifica as faixas de doação

if (doacao < 1000.00):
    investimento = doacao * 0.05
    uso_imediato = doacao - investimento
else:
    investimento = doacao * 0.15
    uso_imediato = doacao - investimento

print("A doação de R$ {:.2f} implica em um investimento de R$ {:.2f},
restando R$ {:.2f} para uso imediato".format(doacao, investimento,
uso_imediato))
```

O formatador `:.2f` fará com que nossas variáveis sejam exibidas sempre com duas casas decimais.

- Temos ainda mais um tipo de *if* para conhecer: o **desvio condicional encadeado** (ou concatenado).

Trata-se do uso **de um desvio condicional dentro do outro**.

Usamos o *if* encadeado quando queremos que um teste lógico só aconteça dependendo do resultado de um teste lógico anterior.

Vamos ver um caso onde ele pode ser utilizado:

Umã ONG resolveu prestar um serviço bem diferente: ela oferece vans para buscar pessoas com qualquer tipo de dificuldade de locomoção para poderem votar.

Para evitar problemas no momento do embarque, porém, você foi convidado a criar um programa que valide a idade dos passageiros: **caso tenham 16 anos ou menos**, não podem votar (e nem embarcar). **Caso tenham entre 16 anos e 18 incompletos**, podem optar por votar ou não. **Caso tenham mais de 18 anos**, devem votar obrigatoriamente.

Crie um script chamado **if_encadeado.py** que receba a idade dos passageiros em potencial e indique se podem embarcar e votar.

- • • • •
- O If Encadeado é o uso de um desvio condicional dentro de outro. Dessa
- + forma, a condição do desvio mais “interno” só será testada dependendo
- + • do resultado das condições do desvio mais “externo”.



```
if (condição booleana):  
    instruções para serem executadas caso a 1ª condição seja  
verdadeira  
  
else:  
    if (condição booleana):  
        instruções para serem executadas caso 1ª a condição seja  
falsa e a 2ª condição seja verdadeira  
  
    else:  
        instruções para serem executadas caso a 1ª condição seja  
falsa e a 2ª condição seja falsa
```




```
print("PROGRAMA VALIDADOR DE PASSAGEIROS")
idade = int(input("Por favor, digite a idade do passageiro: "))


if (idade < 16):
    print("O passageiro não pode votar nem embarcar")
else:
    if(idade>=18):
        print("O passageiro deve votar e pode embarcar")
    else:
        print("O voto do passageiro é opcional e ele pode embarcar")
```

O primeiro if valida se o passageiro tem menos de 16 anos. Caso tenha é exibida a mensagem. Caso não tenha, o segundo if verifica se o passageiro tem mais de 18 anos e, caso tenha, é exibida a mensagem correspondente. Caso não tenha, é exibida a mensagem restante.

Portanto aprendemos que para os casos onde temos apenas uma condição à verificar e apenas um conjunto de instruções para o caso de essa condição ser verdadeira, devemos utilizar um **if simples**.


Para os casos onde temos instruções para uma condição que seja verdadeira e outras instruções para o caso de essa mesma condição ser falsa, devemos utilizar um **if composto**.

E para os casos onde temos diversas condições, cada uma com seu conjunto de instruções, devemos utilizar um **if encadeado**.



MAIS ALGUMAS COMPARAÇÕES

APRENDENDO A COMPARAR
DIFERENTES VALORES



Para variáveis de tipos numéricos, as comparações podem ser feitas da seguinte forma:

OPERADOR	SIGNIFICADO
==	Os dois valores são iguais?
=	Atribuição de valor. CUIDADO : se você utilizar em uma condição, uma variável receberá o valor da outra.
>	O valor da esquerda é maior que o da direita?
<	O valor da esquerda é menor que o da direita?
>=	O valor da esquerda é maior ou igual ao da direita?
<=	O valor da esquerda é menor ou igual ao da direita?
!=	O valor da esquerda é diferente do valor da direita?

Para variáveis do tipo String, porém, há mais um caminho para comparação!

Podemos verificar se um texto está em uma String utilizando o operador *in*.

Exemplo: *“olá” in saudacao* é uma comparação que retorna VERDADEIRO se o texto *“olá”* estiver dentro da variável *saudacao*.

Crie um script chamado *diversas_comparacoes.py* para testarmos alguns desses operadores



```
+ • #0 teste abaixo retorna Falso
    print("Testando o operador == para comparar 15 e 12: {}".format(15==12))
    #0 teste abaixo retorna Verdadeiro
    print("Testando o operador == para comparar 15 e 15: {}".format(15==15))
    #0 teste abaixo retorna Falso
    print("Testando o operador == para comparar 'Teste' e 'Test':
+ {}").format("Teste"=="Test"))
    #0 teste abaixo retorna Verdadeiro
    print("Testando o operador == para comparar 'Teste' e 'Teste':
{}").format("Teste"=="Teste"))
    #0 teste abaixo retorna Vardadeiro
    print("Testando o operador in para verificar se 'Test' está em 'Teste':
+ {}").format("Test" in "Teste"))
    #0 teste abaixo retorna Falso
    print("Testando o operador in para verificar se 'Teste' está em 'Test':
+ {}").format("Teste" in "Test"))
```



PROGRAMANDO EM PYTHON



CONHECENDO OS
OPERADORES LÓGICOS



- Em diversas ocasiões uma única condição é insuficiente para validar algo dentro dos nossos programas.

Nesses casos precisaremos fazer uso de *operadores lógicos* para realizarmos a *conexão* entre as diferentes condições.

Esses operadores lógicos servem para determinar *quais os resultados possíveis* quando as condições forem avaliadas em conjunto.

Existem diversos operadores lógicos, sendo que nem todas as linguagens de programação implementam todos eles.

No caso do python, existem os operadores **não**, **e** e **ou**.

Vamos entender como cada um deles se comporta?

OPERADOR	NOME	SIGNIFICADO
not	NÃO (NOT)	Inverte o estado lógico da condição que acompanhar: verdadeiro vira falso e falso vira verdadeiro.
and	E (AND)	Avalia as duas condições e retorna verdadeiro apenas se ambas forem verdadeiras.
or	OU (OR)	Avalia as duas condições e retorna verdadeiro se pelo menos uma das duas forem verdadeiras.

HORA DE EXERCITAR

Ocasionalmente a ONG para a qual você trabalha recebe doações em dólar e precisa saber qual é o valor em reais. Crie um programa que permita que o usuário digite o valor da doação em dólares, converta esse valor para reais e exiba o resultado na tela.

HORA DE EXERCITAR

Um funcionário da ONG do exercício anterior realiza o trabalho de buscar alimentos diariamente no Mercado Municipal utilizando um carro. É importante que a ONG saiba quantos quilômetros por litro esse carro faz. Crie um programa em que o usuário digite quantos quilômetros o painel do carro mostra no início de uma viagem, quantos quilômetros ele mostra na chegada ao posto de gasolina e quantos litros foram reabastecidos. O programa deve calcular e exibir a média de quilômetros por litro que o veículo faz.

BIBLIOGRAFIA BÁSICA

BEAZLEY, David. *Python Essential Reference*, 2009.

BHARGAVA, ADITYA Y. *Entendendo Algoritmos. Um guia ilustrado para programadores e outros curiosos*. São Paulo: Ed. Novatec, 2017

CORMEN, THOMAS H. et al. *Algoritmos: teoria e prática*. Rio de Janeiro: Elsevier, 2002

COSTA, Sérgio Souza. *Recursividade*. Professor Adjunto da Universidade Federal do Maranhão.

DOWNEY, ALLEN B. *Pense em Python. Pense como um cientista da computação*. São Paulo: Ed. Novatec, 2016

GRANATYR, Jones; PACHOLOK, Edson. *IA Expert Academy*. Disponível em: <https://iaexpert.academy/>

KOPEC, DAVID. *Problemas clássicos de ciência da computação com Python*. São Paulo: Ed. Novatec, 2019

LINDEN, Ricardo. *Algoritmos Genéticos*. 3 edição. Rio de Janeiro: Editora Moderna, 2012

MCKINNEY, WILLIAM WESLEY. *Python para análise de dados. Tratamento de dados com Pandas, Numpy e Ipython*. São Paulo: Ed. Novatec, 2018

BIBLIOGRAFIA BÁSICA

- TENEMBAUM, Aaron M. **Estrutura de Dados Usando C**. Sao Paulo: Makron Books do Brasil, 1995.
- VELLOSO, Paulo. **Estruturas de Dados**. Rio de Janeiro: Ed. Campus, 1991.
- VILLAS, Marcos V & Outros. **Estruturas de Dados: Conceitos e Tecnicas de implementacao**. RJ: Ed. Campus, 1993.
- PREISS, Bruno P. **Estrutura de dados e algoritmos: Padrões de projetos orientados a objetos com Java**. Rio de Janeiro: Editora Campus, 2001.
- PUGA, Sandra; RISSETTI, Gerson. **Lógica de programação e estruturas de dados**. 2016.
- SILVA, Osmar Quirino. **Estrutura de Dados e Algoritmos Usando C. Fundamentos e Aplicações**. Rio de Janeiro: Editora Ciência Moderna, 2007.
- ZIVIANI, N. **Projeto de algoritmos com implementações em pascal e C**. São Paulo: Editora Thomsom, 2002.

OBRIGADO



FIAP

Copyright © 2023 | Professor Dr. Emerson R. Abraham

Todos os direitos reservados. A reprodução ou divulgação total ou parcial deste documento é expressamente proibida sem o consentimento formal, por escrito, do professor/autor.

