



Python II –

Desenvolvendo aplicações web

Caderno de Atividades

Anderson Iwanezuk Thaczuk

Senac

Python II –

Desenvolvendo aplicações web

Caderno de Exercícios

Anderson Iwanezuk Thaczuk

Sumário

Sumário	3
Apresentação do curso	5
Versões.....	6
PEP 8 - Guia de Estilo Para Python	6
Instalação	6
CAPÍTULO 1 – O AMBIENTE E COMANDOS BÁSICOS	8
Atividade 1 – Preparar o ambiente de trabalho	8
Atividade 2 – Instalar e configurar o Visual Studio Code.....	10
Atividade 3 – Acessar o Terminal do Python no VS Code.....	11
Atividade 4 – “Olá Mundo!” para cumprir a tradição	12
CAPÍTULO 2 – LÓGICA PARA PYTHON	14
Atividade 1 – Variáveis e Estruturas de decisão	14
Atividade 2 – Mais estruturas de decisão.....	16
Atividade 3 – Repetições com Enquanto (While)	17
Atividade 4 – Usando ‘For’ para repetições.....	19
Atividade Extra	22
CAPÍTULO 3 – NÚMEROS E FUNÇÕES	23
Atividade 1 – Variáveis e Estruturas de decisão	23
Atividade 2 – Níveis de dificuldade e pontuação.....	24
Atividade 3 – Escolhendo o jogo.....	27
Atividade 4 – Trabalhando com funções	28
Atividade Extra	30
CAPÍTULO 4 – PALAVRAS E FUNÇÕES.....	31
Atividade 1 – Strings e posições	31
Atividade 2 – Criando e editando listas	33
Atividade 3 – Lista externa.....	35
Atividade 4 – Finalizando o Jogo	37
Atividade Extra	42
CAPÍTULO 5 – Conhecendo o Django.....	43
Atividade 1 – Instalando Django no ambiente Virtual	43
Atividade 2 – Ativando o Servidor e a primeira página	45
Atividade 3 – Páginas e templates no Django	48

Atividade 4 – Reaproveitamento de código com blocos e partials.....	53
Atividade Extra	56
CAPÍTULO 6 – Python, Django, Dados e PostgreSQL.....	57
Atividade 1 – Introdução aos dados dinâmicos	57
Atividade 2 – Instalando o banco de dados PostgreSQL	60
Atividade 3 – Acessando o banco de dados com Models	65
Atividade 4 – Configurando e utilizando o Admin e CRUD.....	66
Atividade Extra	72
CAPÍTULO 7 – Customizando o Django	73
Atividade 1 – Organizando os pratos	73
Atividade 2 – Criando outros apps e relacionamentos	77
Atividade 3 – Exibindo imagens com Django	83
Atividade 4 – Construindo um sistema de busca	86
Atividade Extra	88
CAPÍTULO 8 – Autorizações, usuários e refatoração.....	89
Atividade 1 – Configurando usuários e autorizações	89
Atividade 2 – Usuários, cadastro e segurança	91
Atividade 3 – Login, Dashboard e logout	98
Atividade 4 – Formulários e mensagens	101
Atividade Extra	108
CAPÍTULO 9 – Finalizando CRUD e paginação.....	109
Atividade 1 – Apagando pratos e apps	109
Atividade 2 – Edição de dados.....	111
Atividade 3 – Paginação	116
Atividade 4 – Menu dinâmico	119
Atividade Extra	119

Apresentação do curso

O mercado de TI (Tecnologia da Informação) cresce a cada dia, demandando novos conhecimentos dos profissionais que com ela atuam. As linguagens de programação surgiram e evoluem constantemente. Entre elas, a Python, uma tendência, se mantém na lista das tecnologias preferidas.

A linguagem de programação Python, criada em 1991, é uma linguagem mais simples, intuitiva e acessível, com um nível de abstração mais próximo da linguagem humana, facilitando a compreensão e o aprendizado de seu código. Ainda, possibilita a combinação de uma sintaxe clara e direta com ótimos de bibliotecas, módulos e frameworks.

O Python está sendo largamente empregado em soluções web, aplicações de processamento de texto, análise de dados, desenvolvimento web e machine learning. Por ser aplicada para diferentes finalidades, é uma linguagem bastante procurada pelos desenvolvedores, mesmo aqueles que já atuam com outras linguagens de programação.

O mercado de trabalho demanda profissionais que dominem a linguagem de programação. Atualmente, grandes empresas como Google, Amazon, Facebook, utilizam a linguagem de programação Python.

No Brasil, os desenvolvedores Python encontram maiores oportunidades em setores como o de TI, Business Intelligence e marketing digital. Diante desse cenário, o Senac São Paulo, desenvolveu o curso Python - desenvolvendo aplicações web visando possibilitar aos profissionais da área, desenvolver programação web utilizando a linguagem de programação Python.

Este curso visa a seguinte competência, desenvolver aplicações web com Python, para estudantes e profissionais da área de tecnologia da informação e interessados em iniciar na área de desenvolvimento web com Python utilizando framework.

Versões

O Python Software Foundation (PSF) é uma organização independente e sem fins lucrativos que foi criada para gerenciar os direitos autorais a partir da versão 2.1 do Python. Para saber mais sobre a PSF acesse o site <https://www.python.org/psf/>. Utilizaremos a versão 3, que não é compatível com a versão 2 e posteriormente descontinuada.

PEP 8 - Guia de Estilo Para Python

A grande maioria dos programadores Python acaba adotando grande parte da PEP-8 no seu dia a dia. Por isso, a sua leitura é fortemente recomendada.

Versão original em inglês da PEP 8

<https://www.python.org/dev/peps/pep-0008/>

Versão em português da PEP 8

<https://wiki.python.org.br/GuiaDeEstilo>

Instalação

Para utilizar o **Python** é necessário ter instalado a linguagem e um interpretador, compilador e ambiente de desenvolvimento (IDE), dentre as várias opções utilizaremos para este curso o **Microsoft VS Code** – Ambiente desenvolvimento pela Microsoft que apresenta diversos recursos avançados e é gratuito, no desenvolvimento do curso.

Usaremos o **Git/GitHub**, um sistema de controle de versões, usado nos projetos de desenvolvimento de software, para registrar o histórico de edições dos arquivos.

Posteriormente utilizaremos o **Django** que é um framework web Python de alto nível que permite o desenvolvimento de uma variedade de sites desde sistema de gestão de conteúdo, wikis, passando por redes sociais e sites de notícias.

Faremos uso do **PostgreSQL**, um servidor de banco de dados para o armazenamento seguro de informações. Uma ferramenta de código aberto, que implementa a sintaxe de linguagem SQL e roda nos vários sistemas Operacionais.

É recomendável utilizar uma máquina virtual devido a instalação e configuração de vários softwares e serviços.

CAPÍTULO 1 – O AMBIENTE E COMANDOS BÁSICOS

Neste capítulo você vai instalar os softwares para programar em Python e a IDE.

Objetivos:

- Instalar e preparar o ambiente de desenvolvimento.
- Criar comentários com #.
- Imprimir na tela com o comando **print**.
- Pedir informações para o usuário com o comando **input**.

Atividade 1 – Preparar o ambiente de trabalho

Objetivos:

- Preparar o ambiente de trabalho e a IDE do VS Code em um ambiente Windows.

Instalando o Interpretador Python.

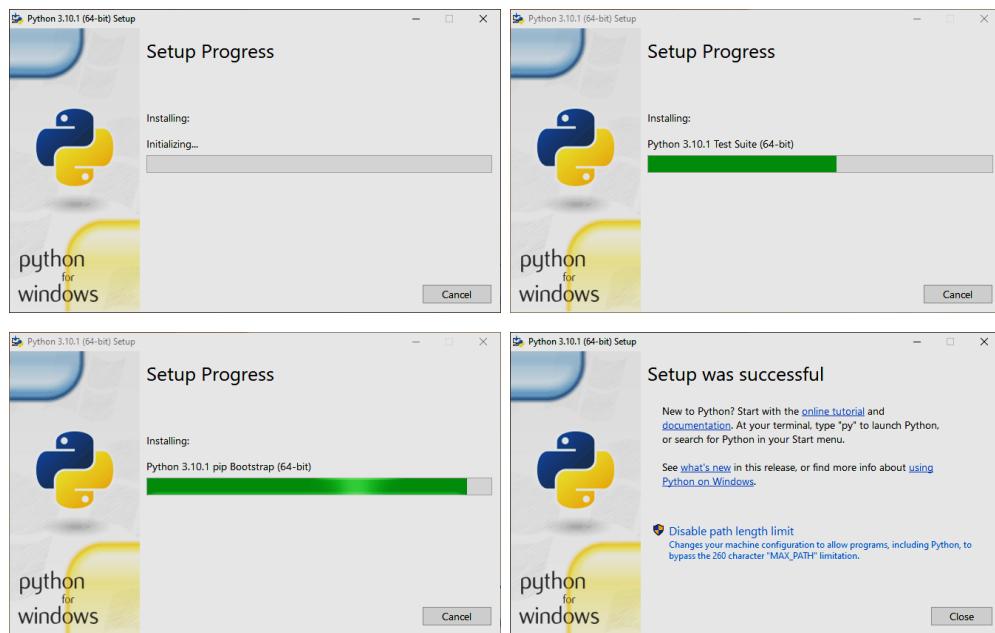
1. Para instalar o interpretador do Python, abra o site: <https://www.python.org/>;



Existem diversas versões do interpretador Python, você vai usar o interpretador oficial da Python Software Foundation, outros interpretadores podem ter um funcionamento um pouco diferente deste, mas os comandos do Python são os mesmos. Instalação de bibliotecas pode ser diferentes de um para o outro.

2. Clique no botão *Downloads* e escolha a versão mais atual do Python.
3. Após o download inicie a instalação.
4. Marque a opção **Add Python 3.X to Path** e clique em **Install Now**.

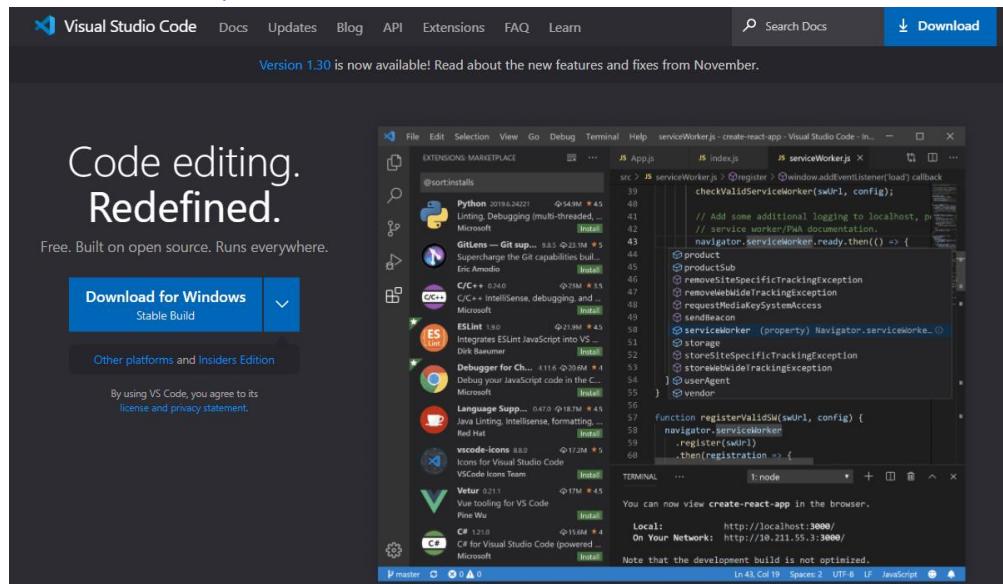




5. Pronto o interpretador já está instalado.

Instalando a IDE

- Para instalar o Ambiente de Desenvolvimento o VS Code, um software da Microsoft totalmente gratuito. abra o site: <https://code.visualstudio.com/>:

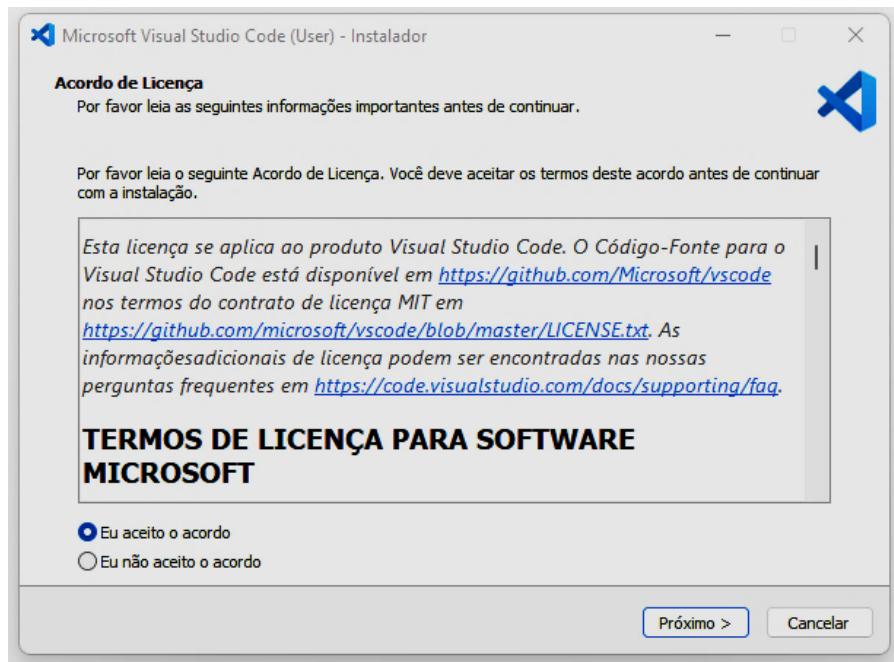


7. Clique do botão de Download.

8. Após o download e inicie a instalação.

9. Marque a opção “Eu aceito o acordo” e continue a instalação, nenhuma outra opção é necessário ser alterada.

10. A Instalação do software é feita em inglês, mas é possível instalar a extensão em português.



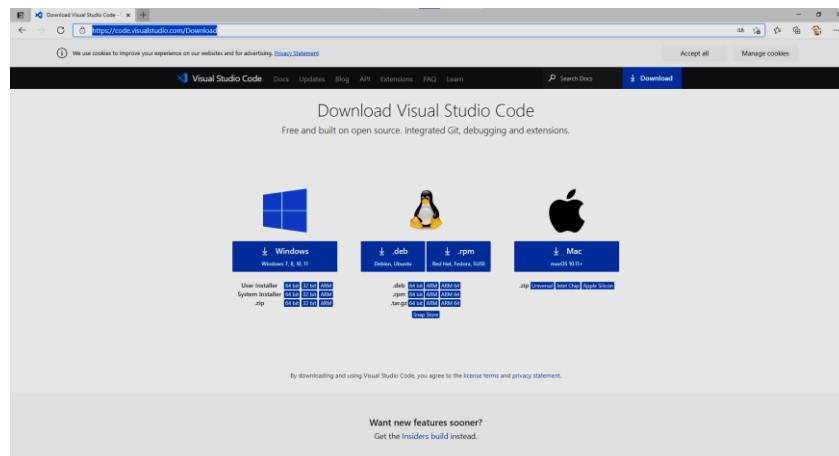
Atividade 2 – Instalar e configurar o Visual Studio Code

Objetivos:

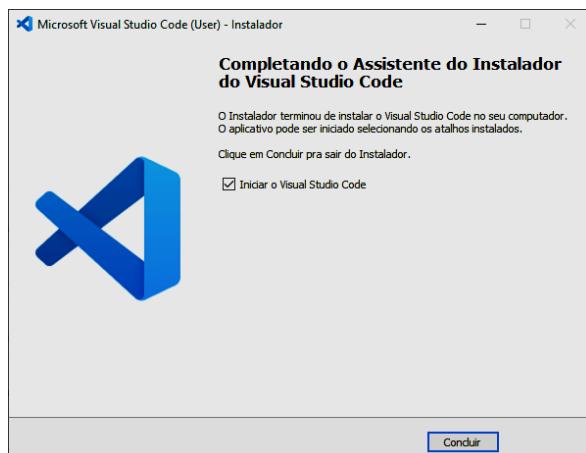
- Instalar e Configurar o Visual Studio Code (VS Code);
- Adicionar extensões para utilização do Python.

Vamos instalar o IDE VS Code e as extensões que alteram o idioma e implementam funcionalidades e ferramentas para ajudar na linguagem de programação que utilizaremos.

1. Acesse <https://code.visualstudio.com/Download> para realizar o Download da versão mais atual do VS Code;



2. Escolha a instalação adequada ao seu sistema operacional.
3. Basta seguir os passos de instalação

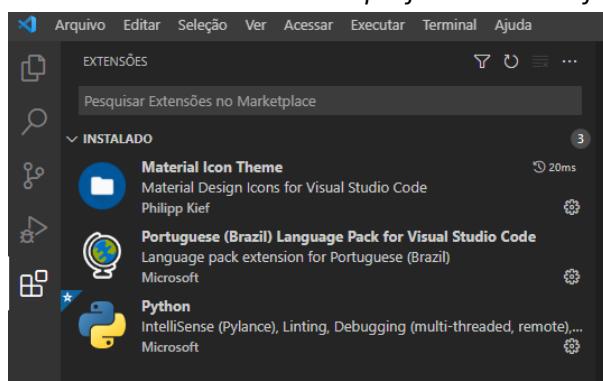


4. Abra o VS Code e na barra lateral procure o botão de Extensão ou pelo menu **Ver / Extensões** (View/Extension).

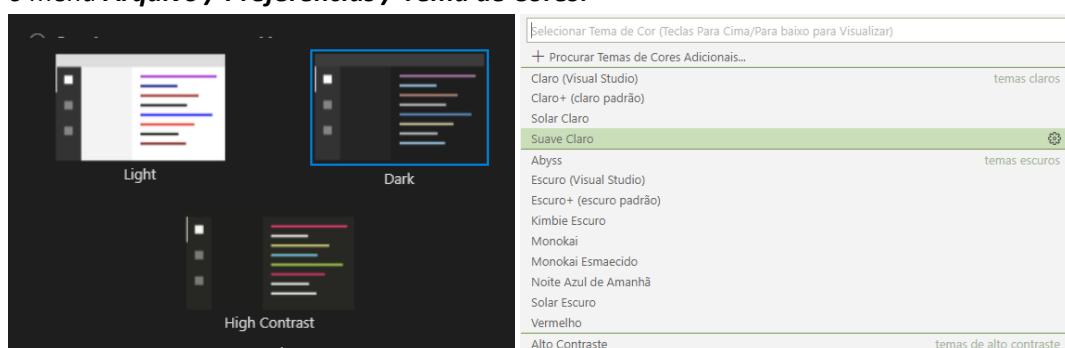


5. Para instalar as extensões basta realizar a pesquisa e clicar em instalar, faça a instalação destas 3 extensões:

- Portuguese (Brazil) Language Pack for Visual Studio Code que altera o idioma do VS Code, será necessário reiniciar o programa para alterar o idioma;
- Python que inclui ferramentas para o desenvolvimento em Python, algumas extensões extras ligadas a linguagem também serão instaladas;
- Material Icon Theme que facilita a identificação dos arquivos do projeto.



6. É possível alterar as cores do ambiente de desenvolvimento, logo após a instalação ou usando o menu **Arquivo / Preferências / Tema de Cores**.



Atividade 3 – Acessar o Terminal do Python no VS Code

Objetivos:

- Acessar o terminal do Python no VS Code;
- Verificar a versão instalada.

Antes de partir para o desenvolvimento da linguagem é preciso verificar se está tudo funcionando e você está com a versão correta do Python.

1. *Abra o VS Code e utilize o menu **Terminal / Novo Terminal** ou **CTRL+ `** para acessar o terminal;*
2. *Digite **python**, aperte Enter.*
3. *Será informado no terminal a versão atual instalado, o indicativo >>> para digitar comandos;*

```
PROBLEMAS SAÍDA CONSOLE DE DEPURAÇÃO TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS C:\Users\anderson.ithaczuk> python
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

4. *Para sair dos comandos digite **exit()**.*

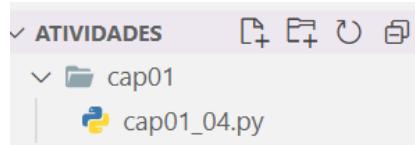
Atividade 4 – “Olá Mundo!” para cumprir a tradição

Objetivos:

- Imprimir na tela “Olá mundo！”, para cumprir a tradição dos programadores ao iniciar o estudo de uma linguagem;
- Revisar os comandos print, input, tipos de dados e format;
- Utilizar o comando input para interagir com o usuário;
- Imprimir a informação capturada em posição de substituição.

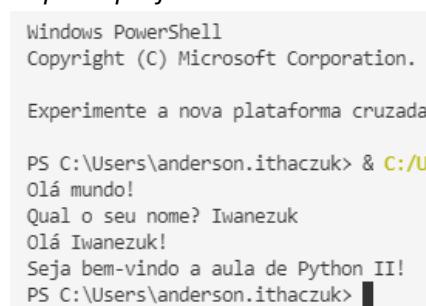
Sempre que vamos aprender uma nova linguagem a primeira atividade em todos os cursos que conhecemos é fazer um “Olá mundo！”; é uma tradição entre os programadores e vamos fazer isso agora:

1. *Vamos criar um diretório para organizar nossos arquivos, crie ‘atividades’, em um local de sua preferência;*
2. *No menu Arquivo / Abrir pasta, abra a pasta recém-criada;*
3. *Crie dentro do VS Code uma nova pasta chamada cap01;*
4. *Abra um novo arquivo no VS Code e salve com o nome de cap1_atividade04.py, dentro do diretório cap01.*



5. *Digite um comentário na primeira linha de código.*
6. *Agora digite o código abaixo para executar no terminal, use o input para pedir o nome do usuário e o print para dizer um 'olá'.*
 - a. **print {} format:** As {} indicam uma posição de substituição que será usada em conjunto com o comando format, conforme exemplo abaixo:
`print('Exemplo de {}'.format('Posição de Substituição'))`

```
# Cumprindo a tradição em Python!!!
print('Olá mundo!')
# podemos usar o comando format para concatenar texto
print('Olá {}' . format(input('Qual o seu nome? ') + '!'))
# é possível colocar dois ou mais textos separados por vírgula
print('Seja bem-vindo a aula de' + ' Python II!')
```

7. *Execute o código usando as teclas **CTRL+F5** ou menu **Executar / Iniciar sem Depuração**. Clicar no botão Executar Arquivo no Terminal, que fica no canto superior direito da tela.*

8. *Observe o resultado do comando no terminal que aparece na parte inferior da tela.*
9. *É mostrado no terminal o caminho do interpretador que está sendo utilizado, o nome do arquivo que foi executado e o resultado do comando print.*

CAPÍTULO 2 – LÓGICA PARA PYTHON

Vamos aprender a declarar e atribuir valores a variáveis em Python, usar estruturas condicionais para tomadas de decisão através de uma condição lógica, com duas ou mais opções.

Trabalharemos com estruturas de repetição para executar uma operação determinadas vezes utilizando os comandos for e while.

Aprenderemos a parar o comando (break) ou continuar (continue) para próxima interação, quando uma condição for testada.

Objetivos:

- Entender o que são as variáveis.
- Trabalhar com os tipos de dados (número, texto).
- Criar estruturas condicionais de decisão if / else.
- Conhecer os operadores básicos de comparação do teste lógico de um if.
- Entender como funciona uma estrutura de loop.
- Utilizar o while para criar um loop e o comando break para terminar a execução.
- Criar loops aninhados usando o for.
- Utilizar o continue para estabelecer uma condição.

Atividade 1 – Variáveis e Estruturas de decisão

Objetivos:

- Revisão uso de variáveis e estruturas de decisão.

Estruturas de decisão seguem um parâmetro lógico, indicando a execução deste ou daquele código e muitas funcionalidades de um software, vamos conhecer melhor esse elemento importante:

1. Crie dentro do VS Code uma nova pasta chamada Atividades/cap02;
2. Salve dentro do diretório criado um novo arquivo como cap02_cap02_01.py.
3. Vamos iniciar apenas informando o objetivo do nosso código e das próximas atividades:

```
print("*****")
print("Adivinhe qual é o Número!")
print("*****")
```

4. Digite o código abaixo, que indicará o número secreto e sua tentativa para tentar adivinhar, qual é.

Note que o valor do número nesse momento será fixo, uma constante e definiremos o seu valor dentro do código, porém tentativa será variável e será digitado através do ‘input’ pelo usuário mudando de valor sempre o código for executado;

```
# Aproveitamos para relembrar constantes e variáveis
numero_secreto = 82

tentativa = input("Tente acertar o número: ")
print("Você digitou: ", tentativa)
```

5. No próximo código já iremos abordar uma estrutura de decisão:

```
se numero_secreto igual tentativa
    escreva("Boa tentativa. ACERTOU!")
senão
    escreva("Não foi dessa vez. ERROU!")
```

6. Para que a estrutura funcione adequadamente precisamos escrever essa instrução em Python:

```
# Aproveitamos para falar de identação (4 espaços ou tab)
# Identação é parte do código Python e faz diferenças em código
if (numero_secreto == tentativa):
    print("Boa tentativa. ACERTOU!")
else:
    print("Não foi dessa vez. ERROU!")
```

7. Você deve executar o código para verificar o correto funcionamento, se ERRAR vai funcionar, e exibir ‘Errou!’, mas se acertar o número também vai exibir ‘ERROU’. Isso ocorre porque o Python entende o valor inserido no input como ‘string’, precisamos converter para inteiro para haja a comparação correta; Observe o corrija o código da forma abaixo:

```
tentativa = input("Tente acertar o número: ")
print("Você digitou: ", tentativa)

# É preciso converter a string pra int para haver comparação correta
# no if
tentativa_int = int(tentativa)

# Aproveitamos para falar de identação (4 espaços ou tab)
# Identação é parte do código Python e faz diferenças em código
if (numero_secreto == tentativa_int):
    print("Boa tentativa. ACERTOU!")
else:
    print("Não foi dessa vez. ERROU!")
```

8. Inclua mais duas linhas ao fim do código. Refaça o teste, e identifique o código de diferentes formas para perceber a influência desse recurso no resultado.

```
print("GAME OVER!")
print('Obrigado por participar!')
```

Atividade 2 – Mais estruturas de decisão

Objetivos:

- Ampliar o conceito de estruturas de decisão.

Muitas vezes uma decisão leva a outra e assim sucessivamente, por isso vamos ampliar o nosso conceito trabalhado anteriormente:

1. *Abra um novo arquivo no VS Code, copie todo o código da atividade anterior e salve no mesmo diretório– com o nome **cap02_02.py**.*
2. *Para ampliar a possibilidade do jogo, caso o usuário não acerte a tentativa, vamos fazer com que ele informe se ele colocou um número maior ou menor, inserindo novas condições dentro das já existentes. Vamos ao código:*

```
# Aproveitamos para falar de identação (4 espaços ou tab)
# Identação é parte do código Python e faz diferenças em código
if (numero_secreto == tentativa_int):
    print("Boa tentativa. ACERTOU!")
else:
    print("Não foi dessa vez. ERROU!")
    # Colocamos novas condições, Lembrando que a identação definirá a
    inserção
    if (tentativa_int > numero_secreto):
        print("Sua tentativa foi MAIOR que o número secreto.")
    if (tentativa_int < numero_secreto):
        print("Sua tentativa foi MENOR que o número secreto.")
    print("GAME OVER!")
print('Obrigado por participar!')
```

3. *Uma boa prática e melhorar a legibilidade do código, para isso vamos tirar as condições do if e inserir em variáveis imediatamente antes da estrutura, e usar como critério;*

```
acerto = tentativa_int == numero_secreto
ehmaior = tentativa_int > numero_secreto
ehmenor = tentativa_int < numero_secreto
# Compara as condições e sempre retorna true ou false

# Aproveitamos para falar de identação (4 espaços ou tab)
# Identação é parte do código Python e faz diferenças em código
if (acerto):
    print("Boa tentativa. ACERTOU!")
else:
    print("Não foi dessa vez. ERROU!")
    # Colocamos novas condições, Lembrando que a identação definirá a
    inserção
    if (ehmaior):
        print("Sua tentativa foi MAIOR que o número secreto.")
```

```

if (ehmenor):
    print("Sua tentativa foi MENOR que o número secreto.")
    print("GAME OVER!")
print('Obrigado por participar!')

```

4. Faça os testes para verificar e corrigir possíveis erros, só então salve o arquivo, feche e finalize a atividade.

Atividade 3 – Repetições com Enquanto (While)

Objetivos:

- Trabalhar com estruturas de repetição While.
- Entender a lógica de um contador.
- Utilizar comando ‘break’ para interromper uma repetição.

Não é nada produtivo ficar copiando códigos, existem estruturas para repetir de maneira muito mais prática e funcional parte da programação, utilizando critérios e formas de execução. Vamos a atividade.

1. Abra o arquivo `cap02_02.py` no VS Code e salve como um novo, nomeando de `cap02_03.py`.
2. Vamos dar uma pequena introdução a biblioteca do sistema operacional para limpar a tela, lembrando que as bibliotecas devem sempre ser declaradas no início do código, em seguida podemos utilizar os recursos importados:

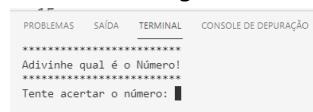
a. `os, system`: A biblioteca que permite acessar informações do sistema operacional é a `os, system` é classe que permite executar os comandos do sistema, você vai usar o comando `cls` que limpa o terminal do Windows em Linux é possível usar o comando `clear`.

```

from os import system
# importa biblioteca do sistema
system('cls') # Limpa tela

```

3. Execute o código e o terminal será exibido com a tela limpa apenas os códigos do Python:



4. Vamos criar uma estrutura para repetir os códigos e possibilitar várias tentativas de acertar o número, utilizando a seguinte lógica:

```

total_de_tentativas = 3

enquanto ainda há total_de_tentativas:
    executa o código

```

5. Vamos construir essa estrutura em python, começando pelas variáveis necessárias ao while:

```
numero_secreto = 82

# Vamos definir a rodada inicial e o total de rodadas
rodada = 1
total_de_tentativas = 3
```

6. Antes da estrutura de repetição vamos colocar o laço de repetição, informando as condições e a mensagem das rodadas realizadas:

- Utilizaremos aqui o {} e o .format para posicionar e indicar as variáveis utilizadas tornando o código mais bem formatado;
- \n indica uma quebra de linha que irá separar apenas visualmente as tentativas;
- {:02d}, indica o uso de 2 dígitos, preenchendo com 'zeros' a esquerda;

```
# O Laço rá englobar todo o código a repetir
while (rodada <= total_de_tentativas):
    print("\nTentativa {:02d} de {:02d}"
          .format(rodada, total_de_tentativas))
```

Mais informações: <https://docs.python.org/3/library/string.html#formatexamples>;

7. Se executarmos o código dessa forma entramos no que chamamos de loop infinito, sendo assim é necessário que se acrescente ao fim da estrutura uma contagem das rodadas, aproveitamos para retirar a linha do print("GAME OVER!"), uma vez que há várias tentativas:

```
rodada = rodada + 1
```

8. Observe o código completo até aqui:

```
from os import system
# importa biblioteca do sistema
system('cls') # Limpa tela

print("*****")
print("Adivinhe qual é o Número!")
print("*****")

# Aproveitamos para relembrar constantes e variáveis
numero_secreto = 82

# Vamos definir a rodada inicial e o total de rodadas
rodada = 1
total_de_tentativas = 3

# O Laço rá englobar todo o código a repetir
while (rodada <= total_de_tentativas):
    print("\nTentativa {:02d} de {:02d}"
          .format(rodada, total_de_tentativas))
# selecione, pressione tab e recue todo o código para indentar dentro
do while
```

```

tentativa = input("Tente acertar o número: ")
print("Você digitou: ", tentativa)
# É preciso converter a string pra int para haver comparação
correta no if
tentativa_int = int(tentativa)

acerto = tentativa_int == numero_secreto
ehmaior = tentativa_int > numero_secreto
ehmenor = tentativa_int < numero_secreto
# Compara as condições e sempre retorna true ou false

# Aproveitamos para falar de identação (4 espaços ou tab)
# Identação é parte do código Python e faz diferenças em código
if (acerto):
    print("Boa tentativa. ACERTOU!\n")
else:
    print("Não foi dessa vez. ERROU!")
    # Colocamos novas condições, Lembrando que a identação
definirá a inserção
    if (ehmaior):
        print("Sua tentativa foi MAIOR que o número secreto.")
    if (ehmenor):
        print("Sua tentativa foi MENOR que o número secreto.")

rodada = rodada + 1

print('\nObrigado por participar!\n')

```

9. Considerando que o correto é que quando o jogador acertar o número não ocorram mais tentativas, vamos inserir um comando para encerrar o laço de repetição:

```

if (acerto):
    print("Boa tentativa. ACERTOU!")
    break

```

10. E para finalizar faça todos os testes possíveis, salve e feche o arquivo.

Atividade 4 – Usando ‘For’ para repetições

Objetivos:

- Trabalhar com estruturas de repetição For.
- Utilizar comando ‘continue’ para seguir para próxima interação

É preciso conhecer várias maneiras de interagir com o código, então vamos explorar outra forma de repetição, o ‘for’:

1. Abra o arquivo `cap02_03` no VS Code e salve como um novo, nomeando de `cap02_04.py`.
2. No uso da estrutura de repetição `for`, temos a definição da variável contadora dentro da própria condição e a contagem é feita através da série ou ‘range’ dos valores, por isso não vamos declarar valor inicial para variável rodada, e vamos apagar a contagem (`rodada = rodada + 1`) ao fim do código:

```
numero_secreto = 82

# Vamos definir apenas o total de tentativas
total_de_tentativas = 3
```

3. Vamos criar a estrutura para repetir os códigos, utilizando o `for`, da seguinte lógica:

```
para variável em série de valores:
    faça algo
```

4. Para utilizar o `for` precisamos adequar a lógica ao código python, entendendo a sintaxe:

- a. `for variável_contadora in range(incremento,valor_final)`

```
# O Laço rá englobar todo o código a repetir
for rodada in range(1, total_de_tentativas):
    print("\nTentativa {:02d} de {:02d}"
          .format(rodada, total_de_tentativas))
# selecione, pressione tab e recue todo o código para identar dentro
do while
    tentativa = input("Tente acertar o número: ")
```

5. Ao executar o código não iremos executar a ‘tentativa 3’, uma vez que o laço se encerra antes, na primeira leitura já é somado um incremento (+1), ao total de tentativas, então é necessária uma pequena correção:

```
# O Laço rá englobar todo o código a repetir
for rodada in range(1, total_de_tentativas + 1):
```

6. Assim como na atividade anterior paramos o código quando acertamos, vamos restringir os números aceitos de 0 a 100, caso não seja digitado o solicitado:

- a tentativa será inválida;
- não será executado a estrutura de decisão logo abaixo;
- não será informado, erro ou acerto;
- uma nova tentativa será iniciada;

7. Vamos definir uma estrutura de decisão e acrescentaremos ‘continue’ como um de seus resultados, que ‘pula’ para próxima interação;

```
for rodada in range(1, total_de_tentativas + 1):
    print("\nTentativa {:02d} de {:02d}"
          .format(rodada, total_de_tentativas))
    tentativa = input("Tente acertar o número de 1 a 100: ")
    print("Você digitou: ", tentativa)

    tentativa_int = int(tentativa)
```

```

if(tentativa_int < 1 or tentativa_int > 100):
    print("Tentativa INVÁLIDA! Somente números de 1 a 100!")
    continue

```

8. Observe o código completo até aqui:

```

from os import system
# importa biblioteca do sistema
system('cls') # Limpa tela

print("*****")
print("Adivinhe qual é o Número!")
print("*****")

# Aproveitamos para relembrar constantes e variáveis
numero_secreto = 82

# Vamos definir apenas o total de tentativas
total_de_tentativas = 3

# O Laço rá englobar todo o código a repetir
for rodada in range(1, total_de_tentativas + 1):
    print("\nTentativa {:02d} de {:02d}"
          .format(rodada, total_de_tentativas))
    # selecione, pressione tab e recue todo o código para indentar dentro
    # do while
    tentativa = input("Tente acertar o número de 1 a 100: ")
    print("Você digitou: ", tentativa)
    # É preciso converter a string pra int para haver comparação
    # correta no if
    tentativa_int = int(tentativa)

    if(tentativa_int < 1 or tentativa_int > 100):
        print("Tentativa INVÁLIDA! Somente números de 1 a 100!")
        continue

    acerto = tentativa_int == numero_secreto
    ehmaior = tentativa_int > numero_secreto
    ehmenor = tentativa_int < numero_secreto
    # Compara as condições e sempre retorna true ou false

    # Aproveitamos para falar de identação (4 espaços ou tab)
    # Identação é parte do código Python e faz diferenças em código
    if (acerto):
        print("Boa tentativa. ACERTOU!")
        break
    else:
        print("Não foi dessa vez. ERROU!")

```

```
# Colocamos novas condições, Lembrando que a identação
# definirá a inserção
if (ehmaior):
    print("Sua tentativa foi MAIOR que o número secreto.")
if (ehmenor):
    print("Sua tentativa foi MENOR que o número secreto.")

rodada = rodada + 1

print('\nObrigado por participar!\n')
```

9. E para finalizar faça todos os testes possíveis, salve e feche o arquivo.

Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extra para rever e aprimorar o que trabalhamos até aqui:

1. Pesquise diagramas e representações d estruturas lógicas que trabalhou para documentar de forma mais assertiva o desenvolvimento do seus projetos.

CAPÍTULO 3 – NÚMEROS E FUNÇÕES

No capítulo que começa vamos dar continuidade a construção de nosso game, e aprender mais sobre o uso de números, cálculos e funções.

Nesse momento vamos tornar o número secreto aleatório para isso vamos importar um módulo, e trabalhar algumas funções, para randomização e arredondamento

Objetivos:

- Importar módulos e usar funções.
- Utilizar um número aleatório, condições e cálculos relacionados
- Carregar arquivos externos como módulos.
- Criar/definir funções.
- Carregar corretamente módulos e funções interna e externamente.

Atividade 1 – Variáveis e Estruturas de decisão

Objetivos:

- Importar módulos e usar funções.
- Gerar um número aleatório

Dando continuidade ao nosso joguinho vamos importar um módulo, então ‘mão na massa’

1. Crie dentro do VS Code uma nova pasta chamada *Atividades/cap03*;
2. Salve dentro do diretório criado um novo arquivo como *cap03/cap03_01.py*.
3. Na linha inicial do código faça a importação do módulo:

```
import random
```

4. Antes de estabelecer o número randômico vamos entender a diferença entre o ‘int’ e o ‘round’, para isso vamos fazer alguns testes:

```
numero_secreto = int(93.9065044957198)
# int retira a parte decimal, round arredonda.
```

```
print(numero_secreto)
```

Resultado = 93

```
numero_secreto = round(93.9065044957198)
# int retira a parte decimal, round arredonda.
```

```
print(numero_secreto)
```

Resultado = 94

5. Vamos inserir uma função que foi importada:

a. 'random' gera um número decimal entre 0 e 1, por isso multiplicamos por 100;

```
numero_secreto = round(random.random() * 100)
# random gera um número decimal entre 0 e 1, por isso * 100

print(numero_secreto)
```

6. Considerando as duas primeiras tentativas entendemos que o código ainda pode ser simplificado, procure estudar e conhecer melhor as classes, módulos ou bibliotecas pois sempre há possibilidades a serem exploradas para facilitar o trabalho:

a. Quer saber mais: <https://docs.python.org/3/library/random.html#module-random>

b. Dica do Luiz Roberto Albano Junior – Senac Jardim Primavera, favor ler a documentação: <https://docs.python.org/3/library/random.html#random.randint>,

7. Vamos ajustar o código ainda mais utilizando um dos exemplos da página do módulo random;

a. 'randrange', gera número aleatório entre 0 e 100.

```
numero_secreto = random.randrange(0,101)

print(numero_secreto)
```

8. Faça os testes necessários, salve e feche o arquivo.

Atividade 2 – Níveis de dificuldade e pontuação

Objetivos:

- Criar um sistema de dificuldade e pontuação.
- Utilizar condições e cálculos relacionados

Vamos codificar para diminuir ou aumentar o nível de dificuldade do jogo:

1. Abra um novo arquivo no VS Code, copie todo o código da atividade anterior e salve no mesmo diretório– com o nome **cap03_02.py**.

2. Para ampliar a possibilidade do jogo, vamos acrescentar níveis, todos os conceitos aqui já foram trabalhados então vamos codificar rapidamente:

a. 'elif' é usado quando temos mais critérios e escolhas dentro de um if

```
numero_secreto = random.randrange(0,101)
total_de_tentativas = 0
pontos = 1000

print("Qual o nível de dificuldade?")
print("(1) Padawan \n(2) Cavalheiro \n(3) Mestre Jedi")
```

```

nivel = int(input("\nDefina o nível: "))

if(nivel == 1):
    total_de_tentativas = 20
elif(nivel == 2):
    #'elif' é usado quando temos mais critérios e escolhas dentro de
    # um if
    total_de_tentativas = 10
else:
    total_de_tentativas = 5

```

3. Feito o sistema de dificuldade vamos estabelecer a pontuação, primeiro vamos inserir o máximo de pontuação junto as variáveis iniciais:

```

numero_secreto = random.randrange(0,101)
total_de_tentativas = 0
pontos = 1000

```

4. A cada tentativa perde a pontuação inicial;

```

pontos_a_perder = int(pontos / total_de_tentativas)
print('Sua pontuação atual:',pontos)

```

5. Vamos modificar o código para informar quando houver o 'acerto':

```

if (acerto):
    print("Boa tentativa. ACERTOU! Fez {} pontos!".format(pontos))
    break

```

6. Não condição de erro, vamos acrescentar um bônus para quando a tentativa for próxima do número secreto, e criar as variáveis para calcular a pontuação:

```

pontos_proximidade = 50 - abs(numero_secreto - tentativa_int)
pontos = (pontos - pontos_a_perder + pontos_proximidade )

```

7. Em seguida vamos estabelecer uma condição para mostrar a pontuação restante durante as rodadas e quando não houve mais tentativas informar o resultado;

```

if(rodada < total_de_tentativas):
    print('Sua pontuação atual:',pontos)
else:
    print("\n-----")
    print("O número secreto era {}".format(numero_secreto))
    print('Sua pontuação FINAL:',pontos)
    print('GAME OVER')
    print("-----")

```

8. Confira o código completo:

```

import random

from os import system

```

```

system('cls') # Limpa tela

print("*****")
print("Adivinhe qual é o Número!")
print("*****")

numero_secreto = random.randrange(0,101)
total_de_tentativas = 0
pontos = 1000

print("Qual o nível de dificuldade?")
print("(1) Padawan \n(2) Cavalheiro \n(3) Mestre Jedi")

nivel = int(input("\nDefina o nível: "))

if(nivel == 1):
    total_de_tentativas = 20
elif(nivel == 2):
    #`elif` é usado quando temos mais critérios e escolhas dentro de
    #um if
    total_de_tentativas = 10
else:
    total_de_tentativas = 5

pontos_a_perder = int(pontos / total_de_tentativas)
print('Sua pontuação atual:',pontos)

for rodada in range(1, total_de_tentativas + 1):
    print("\nTentativa {:02d} de {:02d}"
          .format(rodada, total_de_tentativas))
    tentativa = input("Tente acertar o número de 1 a 100: ")
    print("Você digitou: ", tentativa)
    tentativa_int = int(tentativa)

    if(tentativa_int < 1 or tentativa_int > 100):
        print("Tentativa INVÁLIDA! Somente números de 1 a 100!")
        continue

    acerto = tentativa_int == numero_secreto
    ehmaior = tentativa_int > numero_secreto
    ehmenor = tentativa_int < numero_secreto

    if (acerto):
        print("Boa tentativa. ACERTOU! Fez {} pontos!"
              .format(pontos))
        break
    else:
        pontos_proximidade = 50 - abs(numero_secreto - tentativa_int)

```

```

        pontos = (pontos - pontos_a_perder + pontos_proximidade )
        print("Não foi dessa vez. ERROU!")
        if (ehmaior):
            print("Sua tentativa foi MAIOR que o número secreto.")
        elif (ehmenor):
            print("Sua tentativa foi MENOR que o número secreto.")
        if(rodada < total_de_tentativas):
            print('Sua pontuação atual:',pontos)
        else:
            print("\n-----")
            print("O número secreto era {}".format(numero_secreto))
            print('Sua pontuação FINAL:',pontos)
            print('GAME OVER')
            print("-----")

print('\nObrigado por participar!\n')

```

9. Faça os testes para verificar e corrigir possíveis erros, só então salve o arquivo e feche.
10. Efetue uma cópia do arquivo cap03_02.py, com o nome adivinhe.py, vamos utilizar esse arquivo em outro momento.

Atividade 3 – Escolhendo o jogo

Objetivos:

- Criar um menu inicial para os jogos
- Carregar arquivos externos como módulos.

Não é nada produtivo ficar copiando códigos, existem estruturas para repetir de maneira muito mais prática e funcional parte da programação, utilizando critérios e formas de execução. Vamos a atividade.

1. Efetue uma cópia do arquivo cap03_02.py, com o nome adivinhe.py, vamos utilizar para próxima atividade, apenas comente ou retire as linhas abaixo, para os testes iniciais:

```
from os import system
system('cls') # Limpa tela
```

2. No mesmo diretório crie um arquivo chamado aforca.py, com o seguinte código:

```

print("*****")
print("---- Jogo da forca ----")
print("*****")

print("GG forca")

```

```
print('\nObrigado por participar!\n')
```

3. Crie um arquivo para trabalharmos a escolha do game, salve como: cap03_03.py
 - a. Faremos um meio do usuário escolher qual o game ele gostaria de jogar e com isso exercitamos o import

```
# from os import system
# system('cls') # Limpa tela

print("*****")
print("-- Escolha o seu Game! --")
print("*****")

print("(1) Forca \n(2) Adivinhe o número")

jogo = int(input("Qual vai ser o game? "))

if (jogo == 1):
    print("GG forca")
elif (jogo == 2):
    print("GG adivinhe")
```

4. Para utilizarmos os arquivos do games para jogar é preciso importar:

```
import aforca
import adivinhe
```

5. Uma importação sem o devido tratamento irá resultar em uma execução indesejada:

```
*****
---- Jogo da forca ----
*****
GG forca

Obrigado por participar!

*****
Adivinhe qual é o Número!
*****
Qual o nível de dificuldade?
(1) Padawan
(2) Cavalheiro
(3) Mestre Jedi

Defina o nível: ■
```

6. Isso ocorre pois os arquivos são carregados automaticamente, então vamos transformar os códigos dos arquivos em funções na próxima atividade. Salve e feche o arquivo.

Atividade 4 – Trabalhando com funções

Objetivos:

- Criar/definir os jogos como funções.
- Carregar corretamente módulos e funções interna e externamente.

É preciso conhecer várias maneiras de interagir com o código, então vamos explorar outra forma de repetição, o ‘for’:

1. Abra o arquivo `cap03_03` no VS Code e salve como um novo, nomeando de `cap03_04.py`.
2. Abra o arquivo `aforca.py` para que possamos transformar o código em uma função:
 - a. Para estabelecer uma função insira `def nome_funcao():` no início e idente o código para estar contido nela.

```
def gg():
    print("*****")
    print("---- Jogo da forca ----")
    print("*****")

    print("GG forca")
    print('\nObrigado por participar!\n')
```

3. Vamos realizar o mesmo procedimento com o arquivo `adivinhe.py`:

```
import random

def gg():
    print("*****")
    print("Adivinhe qual é o Número!")
```

segue o restante do código...

4. Agora no arquivo `cap03_04.py`, vamos chamar as funções conforme a escolha do jogador:

```
if (jogo == 1):
    print("GG forca")
    aforca.gg()
elif (jogo == 2):
    print("GG adivinhe")
    adivinhe.gg()
```

5. Realize os testes para acessar os jogos diferentes.

6. Aos expandir os testes e rodar os jogos diretamente, eles não serão mais executados, pois agora dependem da função para “chamar” a execução, porém podemos utilizar uma condição interna do Python para verificar e executar a função caso, ela esteja carregada no ambiente principal (`main`),
 - a. Abra os arquivos `aforca.py` e `adivinhe.py` e insira as linhas abaixo, fora da função e ao fim do código;

```
if(__name__ == "__main__"):
    gg()
```

7. Para finalizar a atividade vamos transformar a escolha do jogo também em uma função, confira o código completo do arquivo:

```
import aforca
import adivinhe
```

```

def escolha_jogo():
    from os import system
    system('cls') # limpa tela

    print("*****")
    print("-- Escolha o seu Game! --")
    print("*****")

    print("(1) Forca \n(2) Adivinhe o número")

    jogo = int(input("Qual vai ser o game? "))

    if (jogo == 1):
        print("GG forca")
        aforca.gg()
    elif (jogo == 2):
        print("GG adivinhe")
        adivinhe.gg()

if(__name__ == "__main__"):
    escolha_jogo()

```

8. *E para finalizar faça todos os testes possíveis, salve e feche o arquivo.*

Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extra para rever e aprimorar o que trabalhamos até aqui:

1. *Reveja os códigos desenvolvidos para o Jogo da Forca e individualmente ou em equipe, procure possíveis melhorias ou a criação de novos jogos utilizando a mesma lógica.*

CAPÍTULO 4 – PALAVRAS E FUNÇÕES

No capítulo que começa vamos a construção de nosso segundo game, e aprender mais sobre o uso de strings, listas, tuplas e funções.

Nesse momento vamos trabalhar com o jogo A forca, rever alguns conceitos e aprofundar outros

Objetivos:

- Utilizar o posicionamento de índices e métodos para manipulação de strings;
- Ampliar o conhecimento de incrementos;
- Criar, armazenar e manipular dados em listas fixas e aleatórias
- Carregar arquivos externos.
- Criar/definir métodos internos e externos com a passagem de parâmetros

Atividade 1 – Strings e posições

Objetivos:

- Utilizar strings para entender o posicionamento de índices;
- Utilizar métodos para manipulação de string

Vamos trabalhar um jogo agora envolvendo palavras, strings e entender melhor como trabalhar com dados.

1. Crie dentro do VS Code uma nova pasta chamada *Atividades/cap04*;
2. Copie o arquivo *aforca.py* para dentro do diretório criado e renomeie como *cap04/cap4_01.py*.
3. Vamos definir uma palavra secreta, para o usuário acertar, e colocar os critérios para o jogo considerar tentativas, enquanto o jogador não “morrer” ou “acertar”:

```
palavra_secreta = "banana"

morreu = False
acertou = False

# enquanto não morreu E não acertou
# enquanto não False
# enquanto(true)
while(not morreu and not acertou):
```

4. Vamos criar um input para inserir uma letra, na tentativa de acertar a palavra, e para tanto vamos fazer com que uma estrutura de repetição, percorra letra a letra, para identificar se existe a tentativa:

- a. Ainda podemos ter problemas em relação a diferença entre letras maiúsculas, espaços antes e depois das palavras, mas essa questão vamos tratar mais adiante:

```
while(not morreu and not acertou):  
  
    tentativa = input("Qual a letra? ")  
  
    for letra in palavra_secreta:  
        if (tentativa == letra):  
            print(tentativa)
```

5. O resultado será o seguinte:

```
*****  
----- Jogo da forca -----  
*****  
Qual a letra? a  
a  
a  
a  
GG forca  
Qual a letra? 
```

6. Vamos colocar um index para contar as posições do texto para depois posicionar as tentativas e a letras descobertas para compor a palavras:

- a. Aproveitamos para simplificar a forma de incremento:

```
while(not morreu and not acertou):  
  
    tentativa = input("Qual a letra? ")  
  
    index = 0  
    for letra in palavra_secreta:  
        if (tentativa == letra):  
            print("Encontrei a letra {} na posição {}".format(letra, index))  
        # index = index + 1  
        index += 1
```

7. O Resultado esperado está logo abaixo:

```
*****  
----- Jogo da forca -----  
*****  
Qual a letra? a  
Encontrei a letra a na posição 1  
Encontrei a letra a na posição 3  
Encontrei a letra a na posição 5  
GG forca  
Qual a letra? 
```

8. Já foi colocada a dificuldade para diferenciar letras maiúsculas e minúsculas, então vamos utilizar um método para deixar todas maiúsculas e eliminar esse problema.

- a. Vale a pena uma visita a documentação para conhecer diversos métodos para strings, disponível em: <https://docs.python.org/3/library/stdtypes.html#string-methods>

```
if (tentativa.upper() == letra.upper()):
```

9. Outra dificuldade recorrente são espaços em branco antes e depois das palavras, vamos usar a função 'strip' para eliminar esse inconveniente:

```
tentativa = input("Qual a letra? ")
tentativa = tentativa.strip()
```

10. Confira o código completo:

```
def gg():
    # def nome_funcao(), cria uma função, basta indentar o código para
    inserir ações
    print("*****")
    print("----- Jogo da forca -----")
    print("*****")

    palavra_secreta = "banana"

    morreu = False
    acertou = False

    while(not morreu and not acertou):

        tentativa = input("Qual a letra? ")
        tentativa = tentativa.strip()

        index = 0
        for letra in palavra_secreta:
            if (tentativa.upper() == letra.upper()):
                print("Encontrei a letra {} na posição {}".format(letra, index))
            # index = index + 1
            index += 1

        print("GG forca")

        print('\nObrigado por participar!\n')

if(__name__ == "__main__"):
    gg()
```

11. Faça os testes para verificar e corrigir possíveis erros, só então salve o arquivo e feche.

Atividade 2 – Criando e editando listas

Objetivos:

- Conhecer mais métodos de manipulação de strings
- Ampliar o conhecimento de incrementos;

- Criar, armazenar e manipular dados em listas;

Vamos codificar para criar uma lista de letras para o jogo:

1. Abra um novo arquivo no VS Code, copie o arquivo da atividade anterior e salve no mesmo diretório – com o nome **cap04_02.py**.
2. Vamos nos aproximar do jogo real, e exibir quantas letras a palavra secreta tem, primeiro de forma manual e mais adiante de modo dinâmico, para isso vamos criar uma lista:

```
palavra_secreta = "banana"
letras_acertadas = ["_", "_", "_", "_", "_", "_"]
```

3. Uma vez que as letras acertadas vão ser armazenadas na lista, vamos utilizar os índices para atualizar a lista e posteriormente exibir a parcial da palavra:

```
index = 0
for letra in palavra_secreta:
    if (tentativa.upper() == letra.upper()):
        letras_acertadas[index] = letra
    index += 1

print(letras_acertadas)
```

4. O Resultado das interações será o seguinte:

```
*****
----- Jogo da forca -----
*****
Qual a letra? a
['_', 'a', '_', 'a', '_', 'a']
Qual a letra? b
['b', 'a', '_', 'a', '_', 'a']
Qual a letra? n
['b', 'a', 'n', 'a', 'n', 'a']
Qual a letra? ■
```

5. Para ajustar a exibição da palavra e o também do chute vamos deslocar o método ‘upper’ para modificar a variável inicial:

```
palavra_secreta = "banana".upper()
```

```
tentativa = tentativa.strip().upper()
```

```
if (tentativa == letra):
```

6. Vamos criar uma condição para que ao cometer erros o jogador vai “perdendo” a vida, até que encerra o game após 07 erros, e caso ele acerte todas as letras

```
if(tentativa in palavra_secreta):
    index = 0
    for letra in palavra_secreta:
        if (tentativa.upper() == letra.upper()):
            letras_acertadas[index] = letra
        index += 1
```

```

else:
    erros += 1

    morreu = erros == 7
    acertou = "_" not in letras_acertadas

    print(letras_acertadas)

```

7. Em seguida vamos colocar as condições para quem vencer ou perder o jogo:

```

if(acertou):
    print("Venceu!")
else:
    print("Perdeu!")

```

8. Agora vamos ajustar a quantidade de letras exibidas para corresponder a palavra secreta, para isso vamos utilizar uma inserção na lista, com 'append':

```

palavra_secreta = "jabuticaba".upper()
letras_acertadas = []

for letra in palavra_secreta:
    letras_acertadas.append("_")

```

9. Podemos também simplificar o código, com um recurso do Python chamado List Comprehension:

```

letras_acertadas = ["_" for letra in palavra_secreta]

```

10. E para finalizar faça todos os testes possíveis, salve e feche o arquivo.

Atividade 3 – Lista externa

Objetivos:

- Criar uma lista aleatória para uso
- Carregar arquivos externos.

Nesse momento vamos criar uma lista para escolha aleatória para o nosso jogo.

1. Efetue uma cópia do arquivo cap04_02.py, com o nome cap04_03.py que vamos utilizar para próxima atividade, aproveite para criar um arquivo, chamado frutas.txt:

abacate	carambola	figo	jenipapo
amora	caju	framboesa	kiwi
ameixa	cereja	graviola	laranja
acerola	cacau	goiaba	limao
abacaxi	caqui	groselia	lima
açai	cupuaçu	guarana	lixia
banana	damasco	jaca	melancia

mamao
melao
maracuja
manga
maça
mexerica
morango
nectarina
pera
pessego
pitanga
pinha
quina
roma
tangerina
uva

2. Vamos abrir esse arquivo para leitura dentro do nosso código e utilizar como uma lista de palavras:

```
arquivo = open("cap04/frutas.txt", "r")
palavras = []

for linha in arquivo:
    linha = linha.strip()
    palavras.append(linha)

arquivo.close()
```

3. Para que a lista em si não tenha espaços ou linhas em branco, vamos tratar palavra a palavra e inserir na lista novamente:

```
arquivo = open("frutas.txt", "r")
palavras = []

for linha in arquivo:
    linha = linha.strip()
    palavras.append(linha)

arquivo.close()
```

4. Para escolher uma palavra aleatoriamente vamos importar o módulo random novamente, e utilizar 'randrange' para sortear uma posição e fazer a escolha:

```
numero = random.randrange(0, len(palavras)+1)

palavra_secreta = palavras[numero].upper()
```

5. E para finalizar faça todos os testes possíveis, salve e feche o arquivo.

Atividade 4 – Finalizando o Jogo

Objetivos:

- Criar/definir métodos internos e externos.
- Realizar a passagem de parâmetros

Seguimos para fase final do nosso jogo, melhorando aspectos do código e criando funções e passando parâmetros.

1. Abra o arquivo cap04_03 no VS Code e salve como um novo, nomeando de cap04_04.py.
2. Para organizar melhor o código e carregar os recursos através de funções vamos isolar partes do programa e transformar em funções, começando pela mensagem de abertura:

```
import random

def imprime_abertura():
    print("*****")
```

```

print("----- Jogo da forca ----- ")
print("*****")

def gg():

    imprime_abertura()

```

3. Você também pode utilizar recursos da IDE, seja VsCode ou PyCharm, ao selecionar um código, clique na 'lampadinha' que solicite a extração do método:



A screenshot of PyCharm's code editor showing a context menu with the 'Extract method' option highlighted. Below the menu, the Python code reads from a file, selects a random word, and initializes a list of underscores for letters. The code is as follows:

```

... arquivo = open("cap04/frutas.txt", "r")
...
for linha in arquivo:
    linha = linha.strip()
    palavras.append(linha)

arquivo.close()

numero = random.randrange(0, len(palavras))

palavra_secreta = palavras[numero].upper()
letras_acertadas = ["_" for letra in palavra_secreta]

```

4. Nomeie o novo método e observe que o código foi movido para parte de baixo do arquivo:



A screenshot of PyCharm showing the variable 'palavra_secreta' being renamed to 'new_func'. A tooltip indicates it can be renamed or visualized. The code is as follows:

```

palavra_secreta = new_func()
letras_acertadas = carrega_palavra_secreta() #ta
Enter para Renomear, Shift+Enter para Visualizar

```

- a. A palavra_secreta só existe dentro da função carrega_palavra_secreta(). Para que ela seja retornada, utilizamos a palavra-chave return;



A screenshot of PyCharm showing the completed function definition for 'carrega_palavra_secreta'. The code is as follows:

```

def carrega_palavra_secreta():
    arquivo = open("cap04/frutas.txt", "r")
    palavras = []

    for linha in arquivo:
        linha = linha.strip()
        palavras.append(linha)

    arquivo.close()

    numero = random.randrange(0, len(palavras))

    palavra_secreta = palavras[numero].upper()
    return palavra_secreta

```

5. Vale lembrar também que uma variável só existe dentro da sua função, então é necessário passar esse parâmetro entre parênteses:

```
letras_acertadas = init_letras_acertadas(palavra_secreta)
```



A screenshot of PyCharm showing the completed function definition for 'init_letras_acertadas'. The code is as follows:

```

def init_letras_acertadas(palavra_secreta):
    # letras_acertadas = ["_" for Letra in palavra_secreta]
    # return letras_acertadas

```

```
    return ["_" for letra in palavra_secreta]
```

6. Agora crie um arquivo novo, chamado `desenhos.py` para inserir algumas funções de desenhos para finalizarmos nosso game:

- O primeiro irá mostrar a força a medida em que o jogador erra as letras;
- O segundo caso seja o vencedor;
- O terceiro mostra a palavra e uma mensagem ao perdedor.
- Dica do Luiz Roberto Albano Junior -, você pode conseguir mais desenhos buscando por ASCII art, ou visite <https://www.asciiart.eu/>

```
def forca(erros):
    print(" _____ ")
    print(" | /   |   ")
    print("     |       ")
    print("     |       ")
    print("     |       ")

    if(erros == 1):
        print("     |     ( )   ")
        print("     |           ")
        print("     |           ")
        print("     |           ")

    if(erros == 2):
        print("     |     ( )   ")
        print("     |     \     ")
        print("     |           ")
        print("     |           ")

    if(erros == 3):
        print("     |     ( )   ")
        print("     |     \|   ")
        print("     |           ")
        print("     |           ")

    if(erros == 4):
        print("     |     ( )   ")
        print("     |     \|/   ")
        print("     |           ")
        print("     |           ")

    if(erros == 5):
        print("     |     ( )   ")
        print("     |     \|/   ")
        print("     |     |     ")
        print("     |           ")

    if(erros == 6):
        print("     |     ( )   ")
        print("     |     \|/   ")
        print("     |     |     ")
        print("     |           ")
```

```

        print(" |     /   ")
if (erros == 7):
    print(" |     ( )   ")
    print(" |     \|/   ")
    print(" |     |   ")
    print(" |     / \   ")

print(" |           ")
print("_|__           ")
print()

def vencedor():
    print("Parabéns, você ganhou!")
    print("      _____      ")
    print("     .'==_==_=_.'      ")
    print("    .-\'\':      /-.      ")
    print("    | (|:.      |) |      ")
    print("    '-|:._      |-'      ")
    print("      \'\\::..      /      ")
    print("      ':.. .:'      ")
    print("          ) (      ")
    print("      _.-' '._      ")
    print("      '-----'      ")

def perdedor(palavra_secreta):
    print("Puxa, você foi enforcado!")
    print("A palavra era {}".format(palavra_secreta))
    print("      _____      ")
    print("     /           \      ")
    print("     /           \      ")
    print("    //           \\\\"      ")
    print(" \  |     XXXX     XXXX  | /      ")
    print("  |  XXXX     XXXX  | /      ")
    print("  |  XXX     XXX  |      ")
    print("  |      |      |      ")
    print("  \  XXX     XXX  /|      ")
    print("   | \  XXX     /|      ")
    print("   |  |      |      |      ")
    print("   |  I  I  I  I  I  I  |      ")
    print("   |  I  I  I  I  I  I  |      ")
    print("   \  _      _/      ")
    print("    \_\      /      ")
    print("      \_\_/_/      ")

```

7. Volte ao arquivo `cap04_04.py` e importe o arquivo criado e insira as funções como abaixo:

```

import random
import desenhos

```

```

else:
    erros += 1
    desenhos.forca(erros)

if(acertou):
    desenhos.vencedor()
else:
    desenhos.perdedor(palavra_secreta)

```

8. Confira o código completo do game abaixo:

```

import random
import desenhos

def gg():
# def nome_funcao(), cria uma função, basta indentar o código para
inserir ações

    imprime_mensagem_abertura()
    palavra_secreta = carrega_palavra_secreta()
    letras_acertadas = init_letras_acertadas(palavra_secreta)
    # reconheceu o método e já passou o parâmetro

    morreu = False
    acertou = False
    erros = 0

    while(not morreu and not acertou):

        tentativa = input("Qual a letra? ")
        tentativa = tentativa.strip().upper()

        if(tentativa in palavra_secreta):
            index = 0
            for letra in palavra_secreta:
                if (tentativa == letra):
                    letras_acertadas[index] = letra
                index += 1
        else:
            erros += 1
            desenhos.forca(erros)

        morreu = erros == 7
        acertou = "_" not in letras_acertadas
        print(letras_acertadas)

    if(acertou):
        desenhos.vencedor()

```

```

        else:
            desenhos.perdedor(palavra_secreta)

    print('\nObrigado por participar!\n')

def imprime_mensagem_abertura():
    print("*****")
    print("----- Jogo da forca -----")
    print("*****")

def carrega_palavra_secreta():
    arquivo = open("cap04/frutas.txt", "r")
    palavras = []

    for linha in arquivo:
        linha = linha.strip()
        palavras.append(linha)

    arquivo.close()

    numero = random.randrange(0, len(palavras))

    palavra_secreta = palavras[numero].upper()
    return palavra_secreta

def init_letras_acertadas(palavra_secreta):
    # Letras_acertadas = ["_" for letra in palavra_secreta]
    # return letras_acertadas
    return ["_" for letra in palavra_secreta]

if(__name__ == "__main__"):
    gg()

```

9. Para finalizar faça todos os testes possíveis, salve e feche o arquivo.

Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extra para rever e aprimorar o que trabalhamos até aqui:

- a. Expanda a possibilidade de criar e utilizar outras Listas de palavras: por dificuldade, por tema etc.

CAPÍTULO 5 – Conhecendo o Django

Aprenderemos a criar uma API com Django, trabalhando com modelos, serializers, views e URLs.

Falando um pouco sobre o propósito do Django que é o desenvolvimento de aplicações web e sites. Lançado em julho de 2005, possui uma estrutura com suporte à virtualização, autenticação e Templates. Sua principal convenção é o **DRY** (*Don't Repeat Yourself*, que significa não seja repetitivo), visa o máximo de proveito do código criado, evitando código duplicado.

O Django utiliza um padrão similar ao MVC **model-view-controller**, chamado de **MTV** (*model-template-view*), onde dividimos a aplicação em camadas ou partes, tornando o código mais organizado e legível. M: Modelo ou Regras de negócio, T: Template Arquivo html que será renderizado pela V: View Equivalente ao controlador.

Objetivos:

- Instalar e configurar o Django em um ambiente virtual e testar o funcionamento do servidor
- Inicializar um projeto e um app Django;
- Realizar as configurações iniciais de um app próprio, carregando páginas html e todos os elementos css, js e outros necessários ao projeto;
- Entender e utilizar elementos estáticos em webpages, com código Python dentro do html;
- Criar estruturas de repetição de código e elementos parciais, que possam ser usados em todo o projeto.

Atividade 1 – Instalando Django no ambiente Virtual

Objetivos:

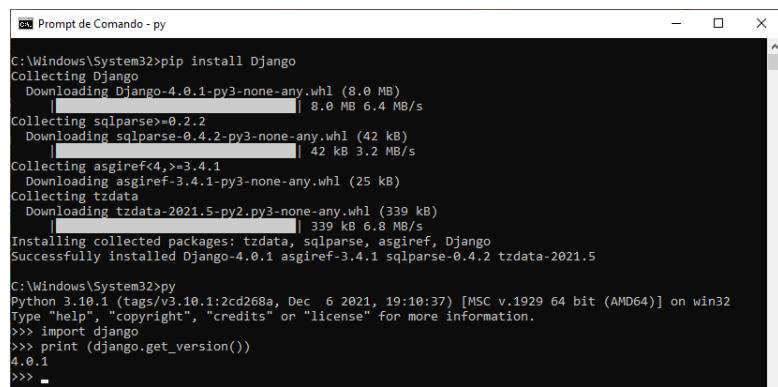
- Instalar o ambiente virtual e Django;
- Executar e testar o funcionamento do servidor

O foco agora é entender o funcionamento e a instalação do Django fique atento pois todos os detalhes são muito importantes:

1. *Vamos instalar o Django para a utilização nas próximas atividades, você pode instalar o Django em todo o sistema operacional, executando no terminal o seguinte comando:*
`pip install Django`
2. *Para testar abrindo o terminal interativo do Python, digitando python ou py no prompt e executando os comandos:*

a. Você também pode acessar o guia de instalação em:
<https://www.djangoproject.com/download/>

```
>>> import django  
>>> print(django.get_version())
```



```
c:\Windows\System32>pip install Django  
Collecting Django  
  Downloading Django-4.0.1-py3-none-any.whl (8.0 MB)  
    8.0 MB 6.4 MB/s  
Collecting sqlparse>=0.2.2  
  Downloading sqlparse-0.4.2-py3-none-any.whl (42 kB)  
    42 kB 3.2 MB/s  
Collecting asgiref<4,>=3.4.1  
  Downloading asgiref-3.4.1-py3-none-any.whl (25 kB)  
Collecting tzdata  
  Downloading tzdata-2021.5-py2.py3-none-any.whl (339 kB)  
    339 kB 6.8 MB/s  
Installing collected packages: tzdata, sqlparse, asgiref, Django  
Successfully installed Django-4.0.1 asgiref-3.4.1 sqlparse-0.4.2 tzdata-2021.5  
c:\Windows\System32>py  
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec  6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import django  
>>> print(django.get_version())  
4.0.1  
>>>
```

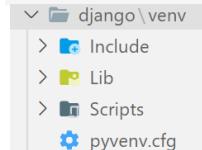
3. Para este curso vamos instalar o Django apenas para uma aplicação, que trabalha com ambiente virtuais, 'VENV', que contêm seus próprios diretórios, arquivos binários, versões e módulos independentes do sistema operacional.

a. Saiba mais em: <https://docs.python.org/3/library/venv.html>

4. Vamos Criar uma diretório para trabalharmos as próximas atividades e armazenar essas configurações: 'django';

5. Acessamos o diretório pelo terminal e digitamos:

```
Cd django  
python -m venv .\venv\
```



6. Em seguida vamos ativar o ambiente virtual com o comando:

a. Utilize um terminal cmd, o powershell pode executar um bloqueio por segurança
venv\Scripts\activate

7. Para saber se o ambiente virtual está ativo repare na palavra venv no início do diretório:

```
C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python -m venv .\venv\  
C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>venv\Scripts\activate  
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>
```

8. Vamos instalar o Django no ambiente virtual:

```
pip install django
```

9. O resultado será o seguinte:

```
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>pip install django
Collecting django
  Using cached Django-4.0.1-py3-none-any.whl (8.0 MB)
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.2-py3-none-any.whl (42 kB)
Collecting asgiref<4,>=3.4.1
  Using cached asgiref-3.5.0-py3-none-any.whl (22 kB)
Collecting tzdata
  Using cached tzdata-2021.5-py2.py3-none-any.whl (339 kB)
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.5.0 django-4.0.1 sqlparse-0.4.2 tzdata-2021.5
WARNING: You are using pip version 21.2.3; however, version 21.3.1 is available.
You should consider upgrading via the 'C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django\venv\Scripts\python.exe -m pip
install --upgrade pip' command.

(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>
```

10. Note a mensagem de advertência solicitando a atualização de versão utilize o comando para atualizar:

```
python -m pip install --upgrade pip
```

11. Digite o comando para verificar os módulos instalados:

```
pip freeze
```

```
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>pip freeze
asgiref==3.5.0
Django==4.0.1
sqlparse==0.4.2
tzdata==2021.5
```

12. A instalação do ambiente virtual e do Django já foram concluídas com êxito.

Atividade 2 – Ativando o Servidor e a primeira página

Objetivos:

- Inicializar um projeto e um app Django;
- Determinar o adequado funcionamento do framework;
- Realizar as configurações iniciais de um app próprio;
- Carregar a primeira página html utilizando Python e Django.

Vamos iniciar o nosso projeto, mas para isso é necessário compreensão do funcionamento do servidor e das configurações iniciais do Django. Foco nas atividades:

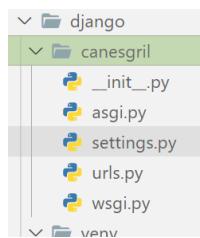
1. Ainda utilizando o terminal vamos iniciar o projeto:

2. Para obter ajuda sobre os comandos do server:

```
django-admin help
```

3. Para iniciar um novo projeto, digite o comando abaixo sem esquecer o 'espaço .' ao final, que serve para criar a pasta de trabalho dentro do diretório principal;

```
django-admin startproject canesgril .
```



4. Vamos observar e configurar alguns dos arquivos que foram criados na pasta 'churrasco', vamos acessar 'settings.py' para configurações:

- A linguagem por padrão utilizamos o 'pt-br';
- O time zone pode ser consultado em:

<https://gist.github.com/mjrulesamrat/0c1f7de951d3c508fb3a20b4b0b33a98>

```
# Internationalization
# https://docs.djangoproject.com/en/4.0/topics/i18n/

LANGUAGE_CODE = 'pt-br'

TIME_ZONE = 'America/Sao_Paulo'
```

5. O arquivo 'urls.py' armazena todas as urls do projeto, como um índice do site.

6. Vamos iniciar o servidor, utilizando os comandos relacionados ao 'manage.py' e no decorrer das atividades vamos conhecendo as opções:

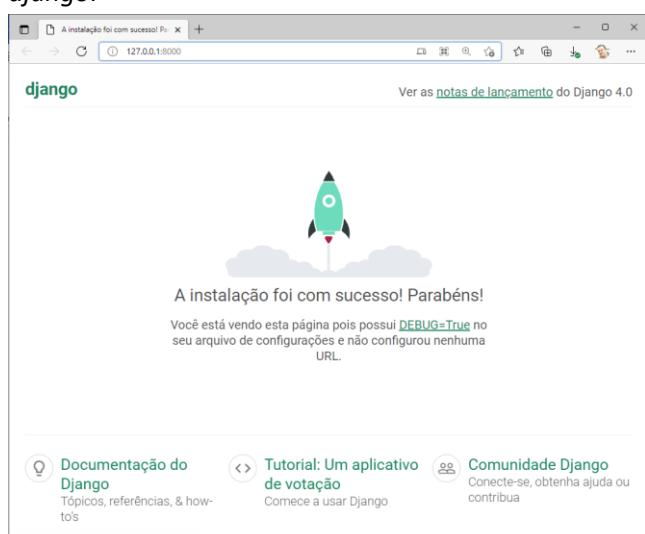
```
python manage.py runserver
```

```
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

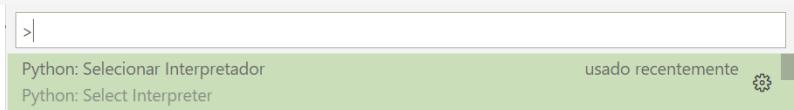
You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations
for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
January 26, 2022 - 00:02:39
Django version 4.0.1, using settings 'canesgril.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

7. Acesse o endereço informado 'http://127.0.0.1:8000/', observe o resultado da instalação django:



8. Vamos configurar e executar o código Python no mesmo ambiente virtual, ou seja, se acionarmos "Ctrl + Shift + P" e digitar:

```
">python select interpreter
```



9. Vamos utilizar o que está em venv. Adicione o caminho no interpretador:



10. Agora subimos um servidor, abrimos outro terminal que carrega automaticamente o venv.

- Um terminal é usado para executar o código ao longo do desenvolvimento da aplicação chamado "2: cmd"
- Outro para executar o próprio servidor chamado "1: server".

11. Vamos iniciar um app com o comando:

- Lembre-se que o 'project' armazena todas as configurações e o app, apenas de app específico

```
python manage.py startapp churras
```

12. Vamos voltar ao arquivo 'canesgril/settings.py' para registrar o app

```
INSTALLED_APPS = [  
    'churras',  
    'django.contrib.admin',  
    ...
```

13. Confira também o arquivo de criação da classe que irá gerenciar nosso app, em 'churras/apps.py':

```
class ChurrasConfig(AppConfig):  
    default_auto_field = 'django.db.models.BigAutoField'  
    name = 'churras'
```

14. Vamos criar um arquivo para exibir nossas URLs de página e não o padrão de instalação do python. Crie um arquivo 'churras/urls.py';

```
from django.urls import path  
  
from . import views  
  
urlpatterns = [  
    path('', views.index, name='index')
```

15. O Django exibiu um erro pois não temos a view 'index', então vamos abrir o arquivo 'churras/views.py' e criar:

```
from django.shortcuts import render
from django.http import HttpResponse

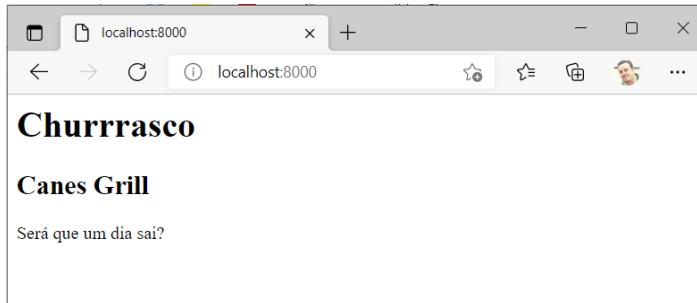
# Create your views here.
def index(request):
    return HttpResponse('<h1>Churrrasco</h1><h2>Canes
Grill</h2><p>Será que um dia sai?</p>')
```

16. Vamos registrar o caminho para o projeto encontrar as páginas do app, em 'canesgril/urls.py':

```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('',include('churras.urls')),
    path('admin/', admin.site.urls),
]
```

17. Após a realização das configurações acesse o localhost:8000 para visualizar a página:



18. Certamente seria complicado desenvolver uma página web dentro do arquivo 'churras/views.py', então nas próximas atividades vamos explorar outras maneiras de exibir nossas páginas.

19. E para finalizar salve e feche os arquivos.

Atividade 3 – Páginas e templates no Django

Objetivos:

- Carregar páginas html e todos os elementos css, js e outros necessários ao projeto;
- Entender e utilizar elementos estáticos em webpages;
- Utilizar código Python dentro do html para elementos estáticos e URLs.

Agora é o momento de aprimorar nossas páginas, com todos os elementos possíveis. Que tal aprender mais sobre Python e Django? Seguimos no aprendizado.

1. Vamos criar uma pasta ‘churras/templates’ para armazenar nossas páginas, dentro da pasta crie um arquivo ‘index.html’, copie o código abaixo para utilizar nesse primeiro momento:

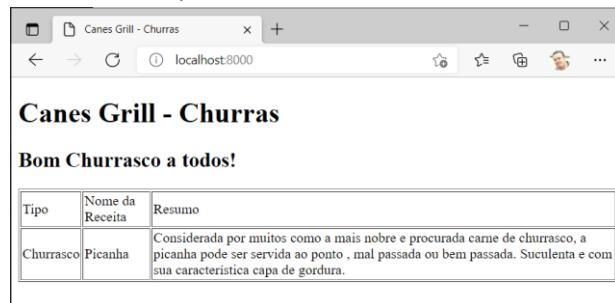
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <title>Canes Grill - Churras</title>
</head>
<body>
    <h1>Canes Grill - Churras</h1>
    <h2>Bom Churrasco a todos!</h2>
    <table border="1">
        <thead>
            <tr>
                <td>Tipo</td>
                <td>Nome do Prato</td>
                <td>Resumo</td>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Churrasco</td>
                <td>Picanha</td>
                <td>Considerada por muitos como a mais nobre e
procurada carne de churrasco, a picanha pode ser servida ao ponto ,
malpassada ou bem passada. Suculenta e com sua característica capa de
gordura.</td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```

2. Vamos abrir o arquivo 'churras/views.py' e alterar a view 'index', para renderizar o arquivo ao invés de ler o código:

```
from django.shortcuts import render

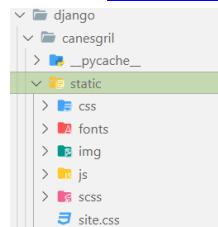
# Create your views here.
def index(request):
    return render(request, 'index.html')
```

3. O resultado espero está abaixo:

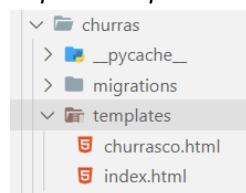


4. Porém vamos melhor e muito o design do meu site, utilizando arquivos html, css e Javascript, para isso é necessário cria um diretório 'canesgril/static', em seguida você irá baixar no material do curso, a pasta cap05/site_churras, descompacte o arquivo e copie todas as pastas e o arquivo 'site.css' para o diretório recém-criado:

- a. Faremos uso de página estáticas e se quiser saber mais pode acessar:
<https://docs.djangoproject.com/pt-br/2.2/howto/static-files/>



5. Copie os arquivos 'churrasco.html' e 'index.html' para a pasta 'churras/templates':



6. Para leitura dos arquivos vamos alterar o arquivo 'canesgril/settings.py', para que encontre os templates, indicando os diretórios que serão utilizados:

```
import os

TEMPLATES = [
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [os.path.join(BASE_DIR, 'churras/templates')],
    ...
}
```

7. Você precisar determinar onde ficarão arquivos estáticos de uso comum, como css, js, imagens e outros, para isso é preciso quase ao fim do arquivo 'canesgril/settings.py', estabelecer, a raiz ou diretório base e o caminho:

```
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
STATIC_URL = 'static/'
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'canesgril/static')
]
...
...
```

8. Vamos coletar as páginas estáticas, acesse o terminal e digite o comando:

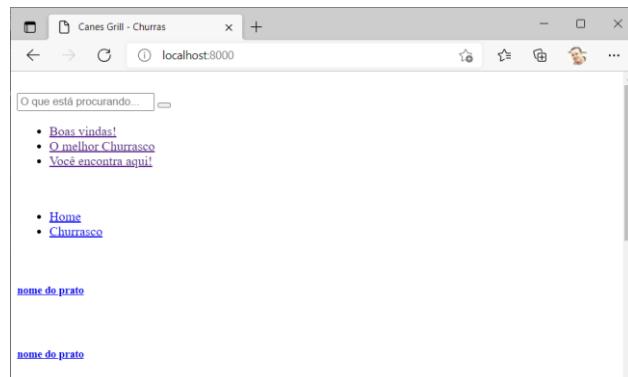
```
python manage.py collectstatic
```

```
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py collectstatic
You have requested to collect static files at the destination
location as specified in your settings.

This will overwrite existing files!
Are you sure you want to do this?

Type 'yes' to continue, or 'no' to cancel: yes
Traceback (most recent call last):
  File "C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django\manage.py", line 22, in <module>
    main()
  File "C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django\manage.py", line 18, in main
```

9. Ao acessar o localhost o resultado ainda não é o esperado:



10. Vamos então corrigir essa questão abrindo o arquivo index.html e inserindo o código python para que ele reconheça os arquivos estáticos, na primeira linha é preciso digitar:

```
{% load static %}
<!DOCTYPE html>
<html lang="pt-br">
...
...
```

11. Todos os links e imagens devem receber a indicação do código python da indicação de arquivo estático como abaixo:

- a. Verifique com calma, salve e visualize a página até todos os elementos estarem disponíveis;

```
<!-- Favicon -->
<link rel="icon" href="{% static 'img/core-img/favicon.ico' %}">

<!-- Stylesheet -->
```

```

<link rel="stylesheet" href="{% static 'site.css' %} ">

<a class="nav-brand" href="index.html"></a>



<!-- jQuery-2.2.4 js -->
<script src="{% static 'js/jquery/jquery-2.2.4.min.js' %}"></script>
<!-- Popper js -->
<script src="{% static 'js/bootstrap/popper.min.js' %}"></script>
<!-- Bootstrap js -->
<script src="{% static 'js/bootstrap/bootstrap.min.js' %}"></script>
<!-- All Plugins js -->
<script src="{% static 'js/plugins/plugins.js' %}"></script>
<!-- Active js -->
<script src="{% static 'js/active.js' %}"></script>

```

12. Agora vamos corrigir os links, mas antes de atuar no html, lembramos que além do 'index.html', temos a página 'churrasco.html', então é preciso acessar o arquivo 'views.py' e criar uma função para renderizar a página:

```

def churrasco(request):
    return render(request, 'churrasco.html')

```

13. Também é preciso incluir os paths nos caminhos padronizados do Django, adicionado código ao arquivo 'urls.py':

```

urlpatterns = [
    path('', views.index, name='index'),
    path('churrasco', views.churrasco, name='churrasco'),
]

```

14. Agora sim vamos corrigir o html:

```

<!-- Logo -->
<a class="nav-brand" href="{% url 'index' %}>
    
</a>

<ul>
    <li><a href="{% url 'index' %}">Home</a></li>
    <li><a href="{% url 'churrasco' %}">Churrasco</a></li>
</ul>

<div class="receipe-content">
    <a href="{% url 'churrasco' %}">
        <h5>nome do prato</h5>

```

```

<!-- Footer Logo -->


<a href="{% url 'index' %}"></a>


```

15. E para finalizar faça todos os testes possíveis, salve e feche o arquivo.

Atividade 4 – Reaproveitamento de código com blocos e partials

Objetivos:

- Evitar o uso repetitivo de códigos;
- Criar estruturas de repetição de código;
- Criar elementos parciais com foco em partes da página, que possam ser usados em todo o projeto.

Ficar copiando, colando e corrigindo o mesmo código por inúmeras vezes não é uma boa prática. Que tal aprendermos como dividir e reutilizar as estruturas de código? Seguimos nessa viagem.

1. Trabalhando ainda com o arquivo ‘index.html’, verificamos que a página de ‘churrasco.html’ teria que ser corrigidas as mesmas questões, porém vamos utilizar outros recursos para não ter que fazer isso e otimizar o nosso código;
2. Crie no diretório templates, um novo arquivo ‘base.html’, e vamos retirar trechos do código de ‘index.html’ para inserir nesse arquivo, os trechos estão logo abaixo:

```

{% load static %}
<!DOCTYPE html>
<html lang="pt-br">

<head>
    <meta charset="UTF-8">
    <meta name="description" content="">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1, shrink-to-fit=no">
        <!-- The above 4 meta tags *must* come first in the head; any
other head content must come *after* these tags -->

        <!-- Título -->
        <title>Canes Grill - Churras</title>

        <!-- Favicon -->
        <link rel="icon" href="{% static 'img/core-img/favicon.ico' %}">

        <!-- Stylesheet -->

```

```

<link rel="stylesheet" href="{% static 'site.css' %}" >

</head>

<body>

    <!-- ##### All Javascript Files ##### -->
    <!-- jQuery-2.2.4 js -->
    <script src="{% static 'js/jquery/jquery-2.2.4.min.js' %}"></script>
    <!-- Popper js -->
    <script src="{% static 'js/bootstrap/popper.min.js' %}"></script>
    <!-- Bootstrap js -->
    <script src="{% static 'js/bootstrap/bootstrap.min.js' %}"></script>
    <!-- All Plugins js -->
    <script src="{% static 'js/plugins/plugins.js' %}"></script>
    <!-- Active js -->
    <script src="{% static 'js/active.js' %}"></script>
</body>

</html>

```

3. Entre os dois códigos do arquivo ‘base.html’ seria inserido os códigos diferenciados das páginas para isso vamos inserir o código python, delimitando onde se inicia e termina os ‘blocos de conteúdo’:

```

<body>

    {% block content %} {% endblock %}

    <!-- ##### All Javascript Files ##### -->

```

4. Voltamos ao arquivo ‘index.html’ e carregamos os arquivos estáticos e a extensão que acabamos de criar, bem como onde o bloco de conteúdo inicia e termina:

```

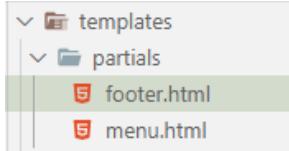
{% extends 'base.html' %}
{% load static %}
{% block content %}
    <!-- Preloader -->

    </footer>
    <!-- ##### Footer Area Start ##### -->
    {% endblock %}

```

5. Podemos utilizar o mesmo recurso na página ‘churrasco.html’, assim não teremos que repetir as correções de código que realizamos e o princípio é idêntico a do item anterior;

6. Para corrigir questões do menu/navegação então vamos utilizar o recurso de ‘partials’, então vamos criar um diretório ‘partials’ dentro de ‘churrasco/templates’, em seguida já criamos os arquivos ‘menu.html’ e ‘footer.html’:



7. Abra o ‘index.html’ e vamos retirar o trecho de código para inserir na página ‘menu.html’:
- Vamos inserir na primeira linha do arquivo ‘menu.html’ pois temos que carregar arquivos estáticos:

```
{% load static %}
```

```
<!-- ##### Header Area Start ##### -->
<header class="header-area">
...
</header>
<!-- ##### Header Area End ##### -->
```

8. No local onde foi retirado o código da página ‘index.html’ vamos inserir o código python:

```
</div>

{% include 'partials/menu.html' %}

<!-- ##### Best Receipe Area Start ##### -->
```

9. O mesmo procedimento deve ser realizado com a página ‘churrasco.html’;

10. Vamos fazer os mesmos procedimentos para o footer. Abra o ‘index.html’ e retire o trecho de código para inserir na página ‘footer.html’:

```
{% load static %}
<!-- ##### Footer Area Start ##### -->
<footer class="footer-area">
...
</footer>
<!-- ##### Footer Area Start ##### -->
```

11. No local onde foi retirado o código da página ‘index.html’ e ‘churrasco.html’ vamos inserir o código python:

```
<!-- ##### Best Receipe Area End ##### -->

{% include 'partials/footer.html' %}

{% endblock %}
```

12. Para finalizar faça todos os testes possíveis, salve e feche o arquivo.

Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extra para rever e aprimorar o que trabalhamos até aqui:

1. *Procure saber um pouco mais sobre o Django e todo o seu Ecossistema, bem como outros frameworks baseados em Python como Flask, por exemplo.*

CAPÍTULO 6 – Python, Django, Dados e PostgreSQL

Agora é hora de trabalharmos com dados, a princípio utilizando listas e dicionários, mas em seguida utilizaremos um banco de dados.

O PostgreSQL (<https://www.postgresql.org/>) é um sistema de banco de dados relacional de objeto de código aberto que usa e estende a linguagem SQL combinada com outros recursos. PostgreSQL remonta a 1986 como parte do projeto na Universidade da Califórnia e tem mais de 30 anos de desenvolvimento ativo na plataforma principal.

O PostgreSQL conquistou uma forte reputação por sua confiabilidade, integridade de dados, conjunto de recursos, extensibilidade e dedicação da comunidade para fornecer soluções inovadoras e de desempenho. O PostgreSQL é executado em todos os principais sistemas operacionais, possui complementos como extensor de banco de dados geoespacial PostGIS.

Objetivos:

- Utilizar dados dinâmicos em páginas html utilizando Python e Django
- Instalar e configurar o banco de dados PostgreSQL;
- Utilizar ‘models’ para criar objetos de banco de dados;
- Criar, configurar e utilizar usuário e o Admin Django
- Criar, Consultar, atualizar e deletar dados através do admin Django;
- Transferir parâmetro através da URL para acessar e obter dados específicos em uma página exclusiva.

Atividade 1 – Introdução aos dados dinâmicos

Objetivos:

- Utilizar dados dinâmicos em páginas html utilizando Python e Django

Vamos melhorar as nossas páginas html e inserir dados dinâmicos através de listas e informações inseridas em objetos, para entender o funcionamento desses elementos e posteriormente utilizar bancos de dados. #boraproduzir

1. *O passo inicial iremos abrir o arquivo ‘churras/templates/index.html’ e deixar apenas um indicativo de nome do prato, apagando os elementos repetitivos:*

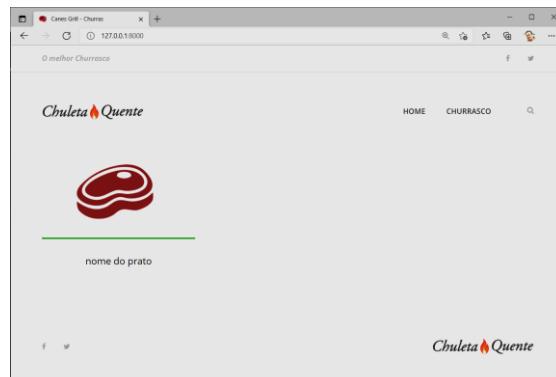
```
{% include 'partials/menu.html' %}
```

```
<!-- ##### Best Receipe Area Start ##### -->
<section class="best-receipe-area">
```

```

<div class="container">
    <div class="row">
        <!-- Single Best Receipe Area -->
        <div class="col-12 col-sm-6 col-lg-4">
            <div class="single-best-receipe-area mb-30">
                
                <div class="receipe-content">
                    <a href="{% url 'churrasco' %}">
                        <h5>nome do prato</h5>
                    </a>
                </div>
            </div>
        </div>
    ...

```



2. Acessamos 'views.py' para ver a função render() e ao incluir mais uma vírgula ao argumento, podemos passar um contexto com informações em formato de dicionário como parâmetro.

```
# Create your views here.
def index(request):
    return render(request, 'index.html', {'prato': 'Costela na brasa'})
```

3. Informaremos no arquivo 'index.html', nosso template onde 'nome_do_prato' deve constar.

- a. Para isso usamos {{}}, e o nome do dicionário.

```
<div class="receipe-content">
    <a href="{% url 'churrasco' %}">
        <h5>{{ prato }}</h5>
    </a>
</div>
```

4. Execute a página para visualizar a inserção dinâmica;



5. Vamos ampliar o dicionário para isso vamos “alimentar” a função com as informações, para isso criamos uma variável com os índices e valores, outra para receber os dados, e no render informamos como parâmetros os dados;

```
def index(request):
    pratos = {
        1:'Picanha',
        2:'Maminha',
        3:'Fraldinha'
    }

    dados = {
        'lista_pratos' : pratos
    }

    return render(request,'index.html',dados)
```

6. Para exibir os dados na página ‘index.html’ iremos repetir a estrutura que exibe o nome dos pratos, utilizando uma estrutura de repetição baseada nos dados renderizados, englobando o código html desejado. Colocamos ‘.items’ ao final do lista_pratos. Atenção ao código:

```
<div class="row">

    {% for chave, prato in lista_pratos.items %}
        <!-- Single Best Receipe Area -->
        <div class="col-12 col-sm-6 col-lg-4">
            <div class="single-best-receipe-area mb-30">
                
                <div class="receipe-content">
                    <a href="{% url 'churrasco' %}">
                        <h5>{{ prato }}</h5>
                    </a>
                </div>
            </div>
        {% endfor %}

    </div>
```

7. O Resultado esperado:

- a. Sugestão: acrescente mais um item ao dicionário para ampliar os resultados.



8. *Salve e feche os arquivos utilizados.*

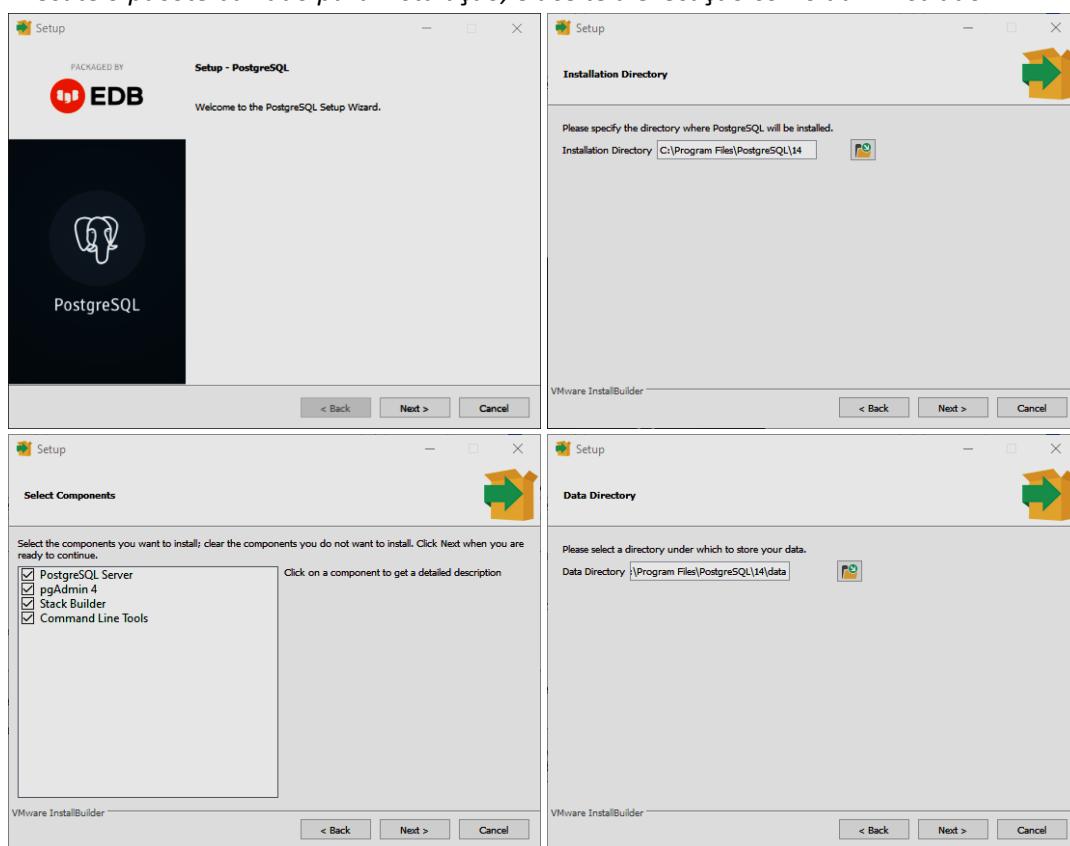
Atividade 2 – Instalando o banco de dados PostgreSQL

Objetivos:

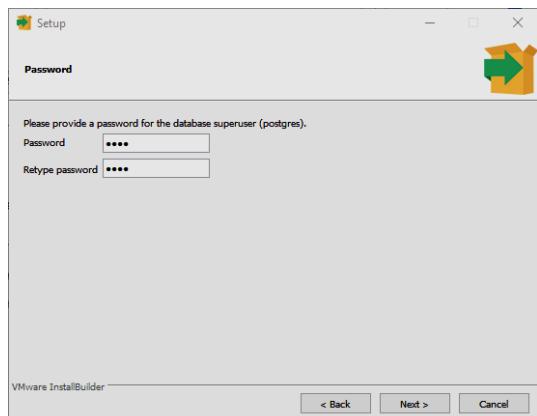
- Instalar e configurar o banco de dados PostgreSQL;

Para as próximas atividades, vamos utilizar o banco de dados PostgreSQL para isso vamos realizar a instalação e configuração. Siga os passos e faça os testes necessários.

1. *Acesse o endereço de download do PostgreSQL e baixe a última versão disponível para instalação no link: <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>*
2. *Execute o pacote baixado para instalação, e aceite a execução como administrador:*



3. *Fique atento a solicitação da senha, ela será necessária para iniciar o PostgreSQL:*
 - a. *Senha para acessar o 1234;*



4. Continue com a instalação:

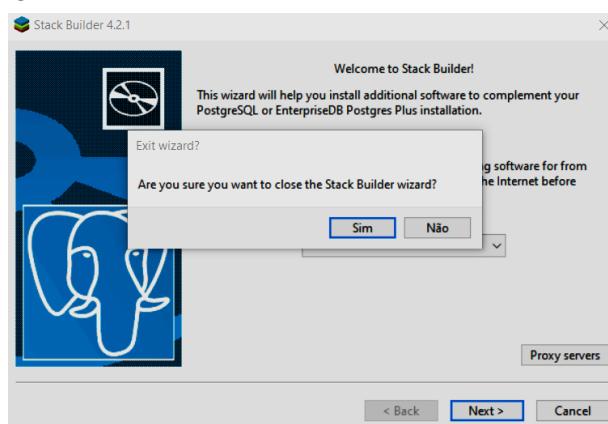
The following settings will be used for the installation:

- Installation Directory: C:\Program Files\PostgreSQL\14
- Server Installation Directory: C:\Program Files\PostgreSQL\14
- Data Directories: C:\Program Files\PostgreSQL\14\data
- Database Ports: 5432
- Database Superuser: postgres
- Operating System Account: NT AUTHORITY\NetworkService
- Database Service: postgresql-x64-14
- Command Line Tools Installation Directory: C:\Program Files\PostgreSQL\14
- pgAdmin4 Installation Directory: C:\Program Files\PostgreSQL\14\pgAdmin 4
- Stack Builder Installation Directory: C:\Program Files\PostgreSQL\14

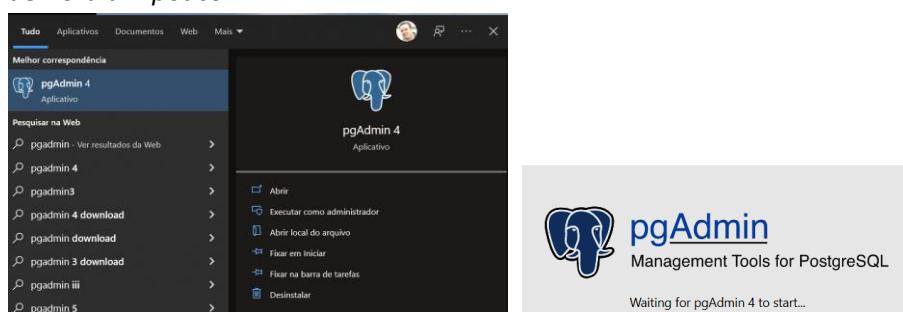
5. Finalize a instalação:



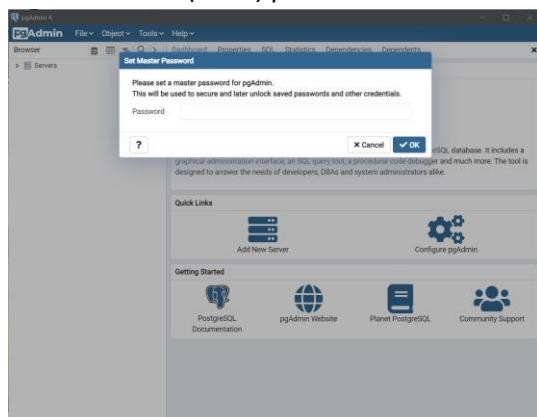
6. Não será necessário realizar as instalações adicionais do ‘Stack Builder’, clique em ‘Cancel’ e ‘Sim’



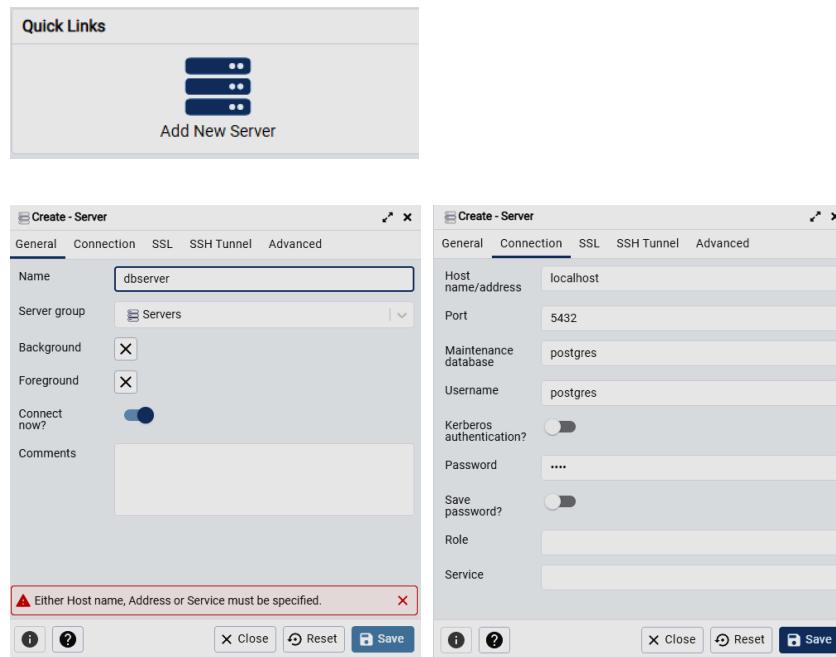
7. Abra o PostgreSQL, buscando no Menu iniciar ‘pgadmin’, Aguarde a primeira inicialização demora um pouco



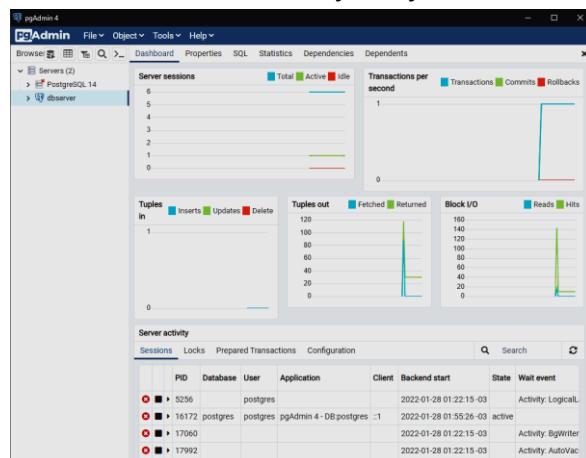
8. Utilize a senha (1234) para o acesso:



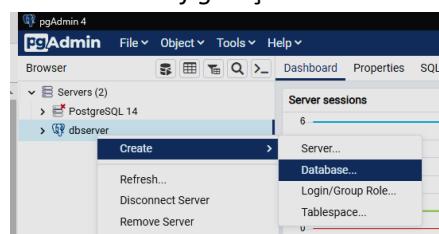
9. Clique em 'Add New Server' em seguida realize as configurações gerais e de conexão conforme as figuras abaixo e utilizando a senha '1234' e finalize salvando;

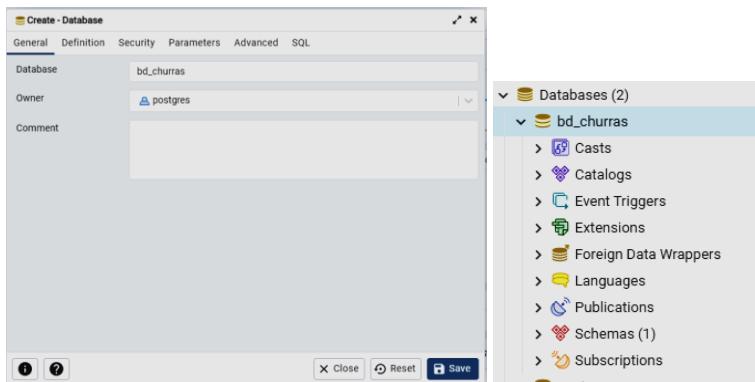


10. Observe o servidor criado e já em funcionamento:



11. Vamos aproveitar para deixar criado o banco de dados 'bd_churras' para futuramente incluir em nossas configurações:





12. Para utilizar qualquer banco de dados no Django é necessário instalar o módulo do respectivo banco para isso vamos executar no terminal o seguinte comando:

```
pip install psycopg2
pip install psycopg2-binary
```

```
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>pip install psycopg2
Collecting psycopg2
  Downloading psycopg2-2.9.3-cp39-cp39-win_amd64.whl (1.2 MB)
    |██████████| 1.2 MB 1.6 MB/s
Installing collected packages: psycopg2
Successfully installed psycopg2-2.9.3

(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>pip install psycopg2-binary
Collecting psycopg2-binary
  Downloading psycopg2_binary-2.9.3-cp39-cp39-win_amd64.whl (1.2 MB)
    |██████████| 1.2 MB 1.3 MB/s
Installing collected packages: psycopg2-binary
Successfully installed psycopg2-binary-2.9.3
```

13. Vamos configurar o uso do PostgreSQL em nosso projeto, acessando 'canesgril/settings.py' e alterando o 'DATABASES':

- a. Se não lembrar das configurações acesse o Postgre, clique com o botão direito em 'dbserver' e visualize a aba 'Connection':



```
# Database
# https://docs.djangoproject.com/en/4.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'bd_churras',
        'USER': 'postgres',
        'PASSWORD': '1234',
        'HOST': 'localhost'
    }
}
```

14. O Banco de dados está instalado e configurado.

Atividade 3 – Acessando o banco de dados com Models

Objetivos:

- Utilizar ‘models’ para criar objetos de banco de dados;

Para acessar e trabalhar com o banco de dados no Django e disponibilizar todas as funcionalidades, faremos uso de ‘models’, assim diminuindo a necessidade de trabalhar com códigos da linguagem SQL e maximizando a produtividade. Vamos as atividades:

1. *Alguns itens são importantes entender antes de iniciar o uso de models e até vale uma pesquisa:*

- a. Um modelo é a única e definitiva fonte de informação sobre seus dados.
- b. Contém os campos e comportamentos dos dados armazenados.
- c. Cada modelo Python é uma subclasse de `models.Model` e cada atributo da classe representa um campo na base de dados;
- d. O Django nos fornece acesso a banco de dados gerado de forma automática, então não precisamos fazer as queries SQL;
- e. Quer saber mais acesse: <https://docs.djangoproject.com/en/3.0/topics/db/models/>

2. Acesse o arquivo ‘churras/models.py’ e construa a classe que irá compor o seu banco de dados:

```
from django.db import models
from datetime import datetime

# Create your models here.
class Prato(models.Model):
    nome_prato = models.CharField(max_length=100)
    ingredientes = models.TextField()
    modo_preparo = models.TextField()
    tempo_preparo = models.IntegerField()
    rendimento = models.CharField(max_length=100)
    categoria = models.CharField(max_length=100)
    date_prato = models.DateTimeField(default=datetime.now,
blank=True)
```

3. Criada a classe, informe que existem dados para serem enviados, execute o código, no terminal:

```
python manage.py makemigrations
```

```
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py makemigrations
Migrations for 'churras':
  churras\migrations\0001_initial.py
    - Create model Prato
```

4. Na lista lateral uma nova pasta chamada "migrations", abra indicado no terminal nesse caso 'churras\migrations\0001_initial.py' para verificar o código criado para migração com as informações passadas ao modelo.
5. Recebemos a informação durante a instalação do Python sobre diversas migrações que são relacionadas ao Django Admin que veremos em breve, e todos os apps registrados em models.py são criados, deletados, editados, visualizados etc., gerando um 'crude' destes modelos. Execute o seguinte código no terminal para realizar esse processo:

```
python manage.py migrate
```

```
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, churras, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying churras.0001_initial... OK
  Applying sessions.0001_initial... OK

(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>
```

6. Após a migração tudo o que é necessário ao Admin tanto quanto a classe e o modelo estão disponíveis. Volte ao PostgreSQL para ver que são criadas várias tabelas do Admin do Django e a nossa churras_prato final. Clicando com o botão direito na tabela, e a opção "View/Edit Data > All Rows" e observamos que um campo id é gerado automaticamente.

The screenshot shows the pgAdmin 4 interface with the database 'bd_churras/postgres@dbserver'. The 'churras_prato' table is selected. The left sidebar shows the table's schema with columns: id, nome_prato, ingredientes, modo_preparo, tempo_preparo, rendimento, categoria, and date_prato. The right pane shows the 'Data Output' tab with the following data:

	id	nome_prato	ingredientes	modo_preparo	tempo_preparo	rendimento	categoria	date_prato
1	[PK] bigint	character varying (100)	text	text	integer	character varying (100)	character varying (100)	timestamp with time zone
2								

7. Nossa banco de dados e a tabela foi criado com sucesso;

Atividade 4 – Configurando e utilizando o Admin e CRUD

Objetivos:

- Criar e configurar o usuário de acesso ao Admin Django
- Configura e utilizar o admin do Django
- Criar, Consultar, atualizar e deletar dados através do admin Django;

- Transferir parâmetro através da URL;
- Acessar e obter dados de um registro específico em uma página exclusiva.

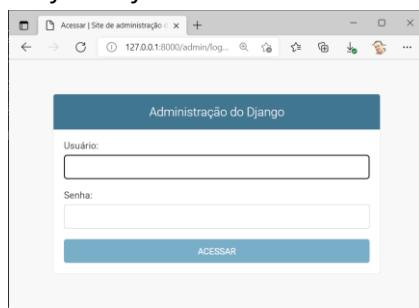
Utilizar a área administrativa de uma aplicação facilita muito a administração de todos os sistemas, mas para isso essa área deve estar configurada e funcional, é o que faremos agora:

1. Abra o arquivo ‘churras/admin.py’ para acessar uma parte integrada ao projeto Django capaz de criar, deletar e editar, vamos registrar nossa classe:

```
from django.contrib import admin
from .models import Prato

# Register your models here.
admin.site.register(Prato)
```

2. Com o servidor ativo, acesse no navegador o endereço ‘http://127.0.0.1:8000/admin’, para verificar o funcionamento do admin:



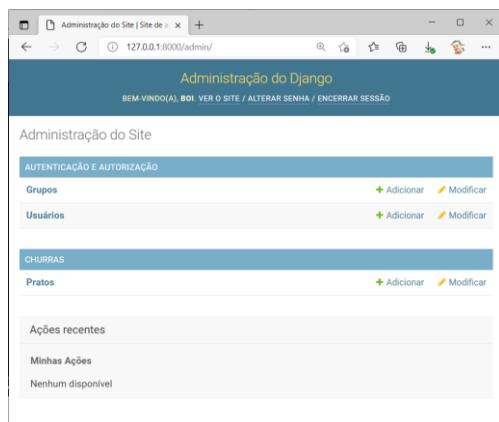
3. Acesse o terminal para criar um usuário que vai acessar o admin como superusuário, digitando o comando:

```
python manage.py createsuperuser
a. Superusuário: boi
b. Email: boi@churras.com
c. Password: 1234
```

```
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py createsuperuser
Usuário (leave blank to use 'iwane'): boi
Endereço de email: boi@churras.com
Password:
Password (again):
Esta senha é muito curta. Ela precisa conter pelo menos 8 caracteres.
Esta senha é muito comum.
Esta senha é inteiramente numérica.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.

(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>
```

4. Volte a tela de login do admin e insira o usuário e senha que acabou de registrar:
 - Faça alguns testes errando a senha;



5. Ao acessar a área de administração do site, note que temos as autenticações e autorizações que trabalharemos mais adiante no curso, e logo abaixo já temos a área do Churras com a Classe pratos, clique em adicionar, para visualizar a cadastrar um novo prato e 'Salvar':
- Note que os campos foram configurados conforme os parâmetros que estabelecemos no 'models'.
 - Nome prato: Picanha
 - Ingredientes: Carvão em brasa, Picanha maturada e Sal Grosso
 - Modo preparo: Considerada por muitos como a mais nobre e procurada carne de churrasco, a picanha pode ser servida ao ponto, mal passada ou bem passada. Suculenta e com sua característica capa de gordura
 - Tempo preparo: 30
 - Rendimento: 3 porções
 - Categoria: Churrasco

Adicionar prato

Nome prato:	<input type="text" value="Picanha"/>
Ingredientes:	Carvão em brasa Picanha maturada Sal Grosso
Modo preparo:	Considerada por muitos como a mais nobre e procurada carne de churrasco, a picanha pode ser servida ao ponto, mal passada ou bem passada. Suculenta e com sua característica capa de gordura
Tempo preparo:	<input type="text" value="30"/>
Rendimento:	<input type="text" value="3"/>
Categoria:	<input type="text" value="Churrasco"/>
Date prato:	Data: <input type="text" value="28/01/2022"/> Hoje <input type="button" value="Calendário"/> Hora: <input type="text" value="03:48:29"/> Agora <input type="button" value="Relógio"/>
<input type="button" value="Salvar e adicionar outro(a)"/> <input type="button" value="Salvar e continuar editando"/> <input type="button" value="SALVAR"/>	

6. Voltamos ao PostgreSQL e clicamos com o botão direito na tabela, selecionando "Refresh", e depois vemos as linhas para conferir se os dados foram salvos.

Data Output							
	id	nome_prato	ingredientes	modo_preparo	tempo_preparo	rendimento	categoria
1	1	Picanha	Carvão em brasa	Considerada por ...	30	3	Churrasco

7. Agora vamos exibir na aplicação os dados que estamos manipulando no banco de dados através do admin. Vamos alterar o dicionário que estávamos usando para uso do banco. Abra o arquivo 'views.py', e modifique para acessar os objetos, ou seja, os pratos, da seguinte forma:

```
from django.shortcuts import render
```

```
from .models import Prato
```

```
# Create your views here.
```

```
def index(request):
```

```
    pratos = Prato.objects.all()
```

```
    dados = {
```

```
        'lista_pratos' : pratos
```

```
}
```

```
    return render(request,'index.html',dados)
```

```
def churrasco(request):
```

```
    return render(request,'churrasco.html')
```

8. Caso ocorra o erro: "Class 'Prato' has no 'objects' member", faça uma busca pelo erro ou acesse <https://stackoverflow.com/questions/45135263/class-has-no-objects-member> e a recomendação é instalar pelo terminal:

```
pip install pylint-django
```

9. Altere também a forma de exibição de dados na página 'index.html', para verificar se a lista está vazia, e os dados para cada prato, da seguinte forma:

```
{% if lista_pratos %}
```

```
{% for prato in lista_pratos %}
```

```
<!-- Single Best Receipe Area -->
```

```
<div class="col-12 col-sm-6 col-lg-4">
```

```
    <div class="single-best-receipe-area mb-30">
```

```
        
```

```
        <div class="receipe-content">
```

```
            <a href="{% url 'churrasco' %}">
```

```
                <h5>{{ prato.nome_prato }}</h5>
```

```
                <h6>{{ prato.categoria }}</h6>
```

```
            </a>
```

```
        </div>
```

```
    </div>
```

```
</div>
```

```
{% endfor %}
```

```
{% else %}
```

```
{% endif %}
```

10. Acesse a área de Admin e cadastre mais um prato para verificar o funcionamento da página html;

- a. Nome prato: Apfelstrudel
- b. Ingredientes: Massa folheada, Maçã, Canela e Sorvete
- c. Modo preparo: Sobremesa tradicional austro-germânica e um delicioso folhado de maçã e canela com sorvete
- d. Tempo preparo: 40
- e. Rendimento: 2 porções
- f. Categoria: Sobremesa



11. O próximo passo é fazer com que ao clicar no prato, ele abra a página 'churrasco.html' com o respectivo prato. Vamos passar os 'ids' dos pratos, para isso abra 'churras/urls.py' e acrescente os códigos:

```
from django.urls import path

from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('<int:prato_id>', views.churrasco, name='churrasco'),
]
```

12. Abra o arquivo 'views.py' para acrescentar os parâmetros de leitura do id:

```
from django.shortcuts import render, get_object_or_404

def churrasco(request, prato_id):
    prato = get_object_or_404(Prato, pk=prato_id)

    prato_a_exibir = {
        'prato' : prato
    }

    return render(request, 'churrasco.html', prato_a_exibir)
```

13. Vamos retirar o link do churrasco do 'menu.html', para evitar o erro de não saber indicar para qual página deve seguir.

14. Na página 'index.html' vamos passar o parâmetro do id como link:

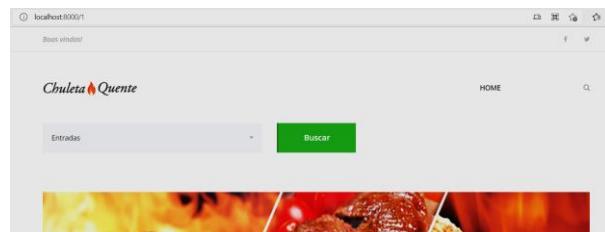
```
<div class="receipe-content">
    <a href="{% url 'churrasco' prato.id %}">
```

```

        <h5>{{ prato.nome_prato }}</h5>
        <h6>{{ prato.categoria }}</h6>
    </a>
</div>

```

15. Note que ao clicar em um prato, é aberta a página do churrasco com a url informando o id:



16. Faça as inserções para que o arquivo 'churrasco.html' exiba dinamicamente os dados dos pratos:

```

<div class="row">
    <div class="col-12 col-md-8">
        <div class="receipe-headline my-5">
            <span>{{prato.date_prato}}</span>
            <h2>{{prato.nome_prato}}</h2>
            <div class="receipe-duration">
                <h6>Preparo: {{prato.tempo_preparo}} minutos</h6>
                <h6>Rendimento: {{prato.rendimento}} porções</h6>
                <h6>Categoria: {{prato.categoria}}</h6>
            </div>
        </div>
    </div>
</div>

<div class="row">
    <div class="col-12 col-lg-8">
        <!-- Single Preparation Step -->
        <div class="single-preparation-step d-flex">
            <p>{{prato.modo_preparo}}</p>
        </div>
    </div>
</div>

<!-- Ingredientes -->
<div class="col-12 col-lg-4">
    <div class="ingredients">
        <h4>Resumo</h4>
        <div class="ingredients">
            <p>{{prato.ingredientes}}</p>
        </div>
    </div>
</div>

```

17. Para finalizar faça todos os testes possíveis, salve e feche os arquivos.

Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extra para rever e aprimorar o que trabalhamos até aqui:

1. *Faça uma pesquisa sobre os principais banco de dados utilizado por desenvolvedores web e com é realizada a integração com o Django.*

CAPÍTULO 7 – Customizando o Django

Já evoluímos muito no Django, instalamos e utilizamos o banco de dados, configuramos acessamos área de administração, porém precisa ser melhorada exibindo os nomes dos pratos, ordem e outros elementos para facilitar o trabalho de quem vai administrar o projeto.

Para isso vamos customizar diversas opções do Django e entender melhor o funcionamento de buscas, filtros e paginações, para que tanto quem vai administrar quanto quem vai acessar tenha uma experiência do usuário satisfatória.

Objetivos:

- Customizar elementos da área administrativa;
- Criar apps e estabelecer relacionamento no banco de dados;
- Criar filtros para visualização de dados;
- Configurar e realizar múltiplas edições de dados;
- Inserir, exibir e alterar imagens na aplicação;
- Criar e configurar um sistema de busca e exibição de dados.

Atividade 1 – Organizando os pratos

Objetivos:

- Customizar elementos da área administrativa

Customizaremos aspectos da Administração da aplicação, criar filtros e paginações;

1. *Observe que ao acessarmos a página do admin que lista os pratos, é complicado saber qual é o prato que está listado, uma vez que o Django delimita como objetos:*

The screenshot shows the Django Admin interface for the 'Pratos' model. The top navigation bar includes links for 'BEM-VINDO(A)', 'BOI', 'VER O SITE', 'ALTERAR SENHA', and 'ENCERRAR SESSÃO'. Below the navigation, there's a breadcrumb trail: 'Início > Churras > Pratos'. A search bar is followed by a 'Start typing to filter...' placeholder. On the left, a sidebar lists 'AUTENTICAÇÃO E AUTORIZAÇÃO' with 'Grupos' and 'Usuários' options, and 'CHURRAS' with 'Pratos' selected. The main content area is titled 'Selecionar prato para modificar'. It contains a dropdown menu for 'Ação:' with '-----' selected, a 'Ir' button, and a message '0 de 2 selecionados'. Below this, there are three checkboxes: 'PRATO', 'Prato object (2)', and 'Prato object (1)'. At the bottom, a summary shows '2 pratos'. A 'ADICIONAR PRATO +' button is located at the top right of the main content area.

2. *Acesse 'admin.py' para criar uma classe e passar um argumento, para editar o 'display' dos pratos:*

```
from django.contrib import admin
from .models import Prato

# Register your models here.
```

```

class ListandoPratos(admin.ModelAdmin):
    list_display =
('id', 'nome_prato', 'categoria', 'tempo_preparo', 'rendimento')

admin.site.register(Prato, ListandoPratos)

```

Ação:	ID	NOME PRATO	CATEGORIA	TEMPO PREPARO	RENDIMENTO
<input type="checkbox"/>	2	Apfelstrudel	Sobremesa	40	2
<input type="checkbox"/>	1	Picanha	Churrasco	30	3

2 pratos

3. Observe que o resultado é satisfatório, porém para acessar o prato temos que clicar no 'id', vamos melhor esse aspecto, acrescente mais uma linha a classe, assim habilitamos o link também para o nome do prato.

```

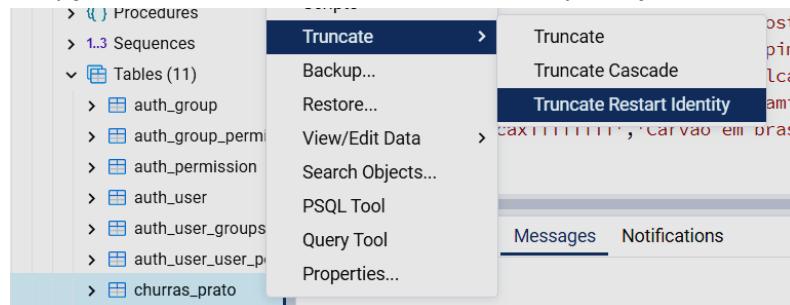
class ListandoPratos(admin.ModelAdmin):
    list_display =
('id', 'nome_prato', 'categoria', 'tempo_preparo', 'rendimento')
    list_display_links = ('id', 'nome_prato')

```

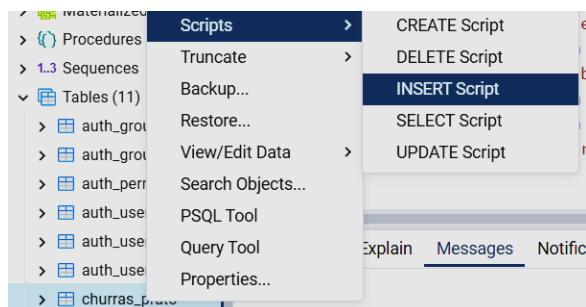
Ação:	ID	NOME PRATO	CATEGORIA	TEMPO PREPARO	RENDIMENTO
<input type="checkbox"/>	2	Apfelstrudel	Sobremesa	40	2
<input type="checkbox"/>	1	Picanha	Churrasco	30	3

2 pratos

4. Para haver um número razoável de registros para os próximos testes vamos, copiar e executar o código sql dentro do postgres, para isso clique com o botão direito no banco 'churras', escolhas as opções 'Truncate >> Truncate Restart Identity', confirme com 'Ok'; Isso irá 'limpar' a tabela.



5. Logo após novamente com o botão direito 'Scripts >> INSERT Script', e na janela que irá abrir cole o código abaixo e execute:



```

INSERT INTO public.churras_prato(
    nome_prato, ingredientes, modo_preparo, tempo_preparo,
rendimento, categoria, date_prato)
VALUES
    ('Picanha','Carvão em brasa, Picanha maturada e Sal Grosso','Considerada por muitos como a mais nobre e procurada carne de churrasco, a picanha pode ser servida ao ponto, malpassada ou bem passada. Suculenta e com sua característica capa de gordura',30,3,'Churrasco','2022-01-30 21:04:31-03'),
    ('Apfelstrudel','Massa folheada, Maçã, Canela e Sorvete','Sobremesa tradicional austro-germânica e um delicioso folhado de maçã e canela com sorvete',40,2,'Sobremesa','2022-01-30 21:05:50-03'),
    ('Picanha ao alho','Carvão em brasa, Picanha maturada, alho e Sal Grosso','Esta é a combinação do sabor inconfundível da picanha com o aroma acentuado do alho. Condimento que casa perfeitamente com este corte nobre', 40,2,'Churrasco','2022-01-30 21:05:50-03'),
    ('Fraldinha','Carvão em brasa, Fraldinha e Sal Grosso','Uma das carnes mais suculentas do cardápio. Requintada, com maciez particular e pouca gordura, e uma carne que chama atenção pela sua textura', 40,2,'Churrasco','2022-01-30 21:05:50-03'),
    ('Costela','Carvão em brasa, Costela e Sal Grosso','A mais procurada! Feita na churrasqueira ou ao fogo de chão, e preparada por mais de 08 horas para atingir o ponto ideal e torná-la a referência de nossa churrascaria', 40,2,'Churrasco','2022-01-30 21:05:50-03'),
    ('Cupim', 'Carvão em brasa, Cupim e Sal Grosso', 'Uma referência especial dos paulistas. Bastante gordurosa e macia, o cupim é uma carne fibrosa, que se desfia quando bem-preparada ', 40,2,'Churrasco','2022-01-30 21:05:50-03'),
    ('Alcatra','Carvão em brasa, Alcatra e Sal Grosso','Carne com muitas fibras, porém macia. Sua lateral apresenta uma boa parcela de gordura. Equilibrando de forma harmônica maciez e fibras.', 40,2,'Churrasco','2022-01-30 21:05:50-03'),
    ('Maminha','Carvão em brasa, Maminha e Sal Grosso','Vem da parte inferior da Alcatra. É uma carne com fibras, porém macia e saborosa.', 40,2,'Churrasco','2022-01-30 21:05:50-03'),
    ('Abacaxiiiiii','Carvão em brasa, Abacaxi e Canela','Abacaxi assado com canela ao creme de leite condensado ', 40,2,'Sobremesa','2022-01-30 21:05:50-03')
;

```

6. Ainda trabalhando no arquivo 'admin.py' vamos fazer um sistema de busca dos pratos, para isso, acrescente mais uma linha ao código, com uma lista;
- Se for acrescentado apenas um valor, ocorre um erro, por isso acrescentamos uma 'vírgula' para dar a ideia de continuidade da lista.

```
class ListandoPratos(admin.ModelAdmin):
    list_display = ('id', 'nome_prato', 'categoria', 'tempo_preparo',
'rendimento')
    list_display_links = ('id', 'nome_prato')
    search_fields = ('nome_prato',)
```

Selecionar prato para modificar

ADICIONAR PRATO +

Ação:	ID	NOME PRATO	CATEGORIA	TEMPO PREPARO	RENDIMENTO
<input type="checkbox"/>	9	Abacaxiiiiiiii	Sobremesa	40	2
<input type="checkbox"/>	7	Alcatra	Churrasco	40	2
<input type="checkbox"/>	2	Apfelstrudel	Sobremesa	40	2
<input type="checkbox"/>	5	Costela	Churrasco	40	2
<input type="checkbox"/>	6	Cupim	Churrasco	40	2
<input type="checkbox"/>	4	Fraldinha	Churrasco	40	2
<input type="checkbox"/>	8	Maminha	Churrasco	40	2
<input type="checkbox"/>	1	Picanha	Churrasco	30	3
<input type="checkbox"/>	3	Picanha ao alho	Churrasco	40	2

9 pratos

7. Incrementando ainda mais o admin, vamos acrescentar um filtro por 'categoria' e a paginação quando houver um número elevado de registros. Para isso, acrescente mais linhas a classe que estamos trabalhando:

```
class ListandoPratos(admin.ModelAdmin):
    list_display =
('id', 'nome_prato', 'categoria', 'tempo_preparo', 'rendimento')
    list_display_links = ('id', 'nome_prato')
    search_fields = ('nome_prato',)
    list_filter = ('categoria',)
    list_per_page = 3
```

Selecionar prato para modificar

ADICIONAR PRATO +

FILTRO

Por categoria

Todos
Churrasco
Sobremesa

Ação:	ID	NOME PRATO	CATEGORIA	TEMPO PREPARO	RENDIMENTO
<input type="checkbox"/>	8	Maminha	Churrasco	40	2
<input type="checkbox"/>	1	Picanha	Churrasco	30	3
<input type="checkbox"/>	3	Picanha ao alho	Churrasco	40	2

1 2 3 9 pratos Mostrar tudo

8. Faça todos os testes possível, salve e feche os arquivos utilizados.

Atividade 2 – Criando outros apps e relacionamentos

Objetivos:

- Criar apps e estabelecer relacionamento no banco de dados;
- Criar filtros para visualização de dados;
- Configurar e realizar múltiplas edições de dados;

Vamos criar um app de pessoas para indicar de quem são os pratos que serão exibidos.

1. Acesse o terminal do python e digite o comando para criar um outro app:

```
py manage.py startapp pessoas
```

2. Abra o arquivo ‘canesgril/settings.py’ para incluir ‘pessoas’ em apps instalados:

```
# Application definition
INSTALLED_APPS = [
    'pessoas',
    'churras',
    'django.contrib.admin',
```

3. Vamos criar algumas ‘pessoas’ na aplicação. Acesse ‘pessoas/models.py’, e crie a classe como o código:

```
from django.db import models

# Create your models here.
class Pessoa(models.Model):
    nome = models.CharField(max_length=200)
    email = models.CharField(max_length=200)
```

4. Vamos registrar as ‘pessoas’ para que o Admin possa manipular esses dados. Acesse ‘pessoas/admin.py’, e registre o modelo como o código:

```
from django.contrib import admin
from .models import Pessoa

# Register your models here.
admin.site.register(Pessoa)
```

5. Execute os comandos para marcar e executar a migração:

```
python manage.py makemigrations
python manage.py migrate
```

```
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py makemigrations
Migrations for 'pessoas':
  pessoaas\migrations\0001_initial.py
    - Create model Pessoa

(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, churras, contenttypes, pessoas, sessions
Running migrations:
  Applying pessoas.0001_initial... OK

(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>
```

6. Confirme a migração observando o banco de dados e o admin:

The screenshot shows the Django Admin interface. On the left, there's a sidebar with a tree view. Under 'pessoas_pessoa', it shows 'Columns (3)' with 'id', 'nome', and 'email'. To the right, there are three main sections: 'AUTENTICAÇÃO E AUTORIZAÇÃO' (Groups and Usuários), 'CHURRAS' (Pratos), and 'PESSOAS' (Pessoas). Each section has 'Adicionar' and 'Modificar' buttons.

7. Vamos adicionar uma pessoa:

The first part of the screenshot shows the 'Adicionar pessoa' form with fields for 'Nome' (Churrasqueiro) and 'Email' (churasqueiro@churras.com). The second part shows a success message: 'O pessoa "Pessoa object (1)" foi adicionado com sucesso.' Below it is a list of selected objects: 'PESSOA' and 'Pessoa object (1)', with a note '1 pessoa'.

8. Para melhor aspectos de visualização que aprendemos anteriormente. Abra o arquivo 'pessoas/admin.py' vamos fazer a lista, links, sistema de busca e paginação:

```
from django.contrib import admin
from .models import Pessoa

# Register your models here.
class ListandoPessoas(admin.ModelAdmin):
    list_display = ('id', 'nome', 'email')
    list_display_links = ('id', 'nome')
    search_fields = ('nome',)
    list_per_page = 2

admin.site.register(Pessoa, ListandoPessoas)
```

Selecionar pessoa para modificar

Pesquisar

Ação: Ir 0 de 1 selecionados

ID	NOME	EMAIL
1	Churrasqueiro	churasqueiro@churras.com

1 pessoa

9. Crie mais um usuário para fazer testes e usaremos ele posteriormente:

Adicionar pessoa

Nome:	Garçom
Email:	garcom@churras.com

10. Vamos vincular os pratos as pessoas através dos relacionamentos de informações/tabelas, no que chamamos de 'foreign key' ou chave estrangeira, que irá vincular cada prato a quem criou o prato;

a. Relacionamentos em bancos de dados é um tema que com certeza vale a pena se aprofundar, sugerimos que faça um curso sobre o assunto, mas você pode ter uma noção em <https://docs.djangoproject.com/en/4.0/topics/db/models/#relationships>

11. Para que possamos vincular dados eles têm que ser registrados numa ordem, dessa forma precisamos da pessoa, e depois dos pratos, para isso vamos acessar o Admin e excluir todos os pratos:

Tem certeza?

Tem certeza de que deseja apagar o prato selecionado?

Resumo

- Pratos: 9

Objetos

- Prato: Prato object (9)
- Prato: Prato object (8)
- Prato: Prato object (7)
- Prato: Prato object (6)
- Prato: Prato object (5)
- Prato: Prato object (4)
- Prato: Prato object (3)
- Prato: Prato object (2)
- Prato: Prato object (1)

Sim, eu tenho certeza Não, me leve de volta

Ação: <input type="button"/> Ir 9 de 9 selecionados
<input checked="" type="checkbox"/> ID Remover pratos selecionados
<input checked="" type="checkbox"/> 9 Abacaxiiiiii Sobremesa 40
<input checked="" type="checkbox"/> 8 Maminha Churrasco 40
<input checked="" type="checkbox"/> 7 Alcatra Churrasco 40

12. Configure novamente 'churras/models.py', para importar o módulo 'Pessoa' e vincular 'em cascata' os dados, caso a pessoa seja, excluída, todas os seus pratos também serão.

```
from django.db import models
from datetime import datetime
from pessoas.models import Pessoa

# Create your models here.
class Prato(models.Model):
    pessoa = models.ForeignKey(Pessoa, on_delete=models.CASCADE)
    nome_prato = models.CharField(max_length=100)
    ingredientes = models.TextField()
```

13. Para atualizar o banco de dados, execute novamente os comandos para marcar e executar a migração:

```
python manage.py makemigrations
```

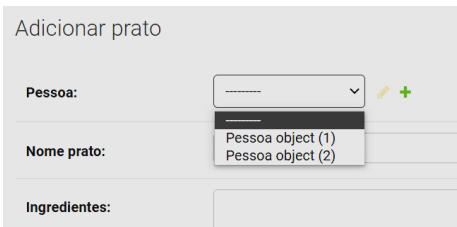
a. para o valor padrão informe '1' e "" (aspas simples vazias)

```
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py makemigrations
needs something to populate existing rows.
Please select a fix:
1) Provide a one-off default now (will be set on all existing rows with a null value for this column)
2) Quit and manually define a default value in models.py.
Select an option: 1
Please enter the default value as valid Python.
The datetime and django.utils.timezone modules are available, so it is possible to provide e.g. timezone.now as a value.
Type 'exit' to exit this prompt
>>> ''
Migrations for 'churras':
  churras\migrations\0002_prato_pessoa.py
    - Add field pessoa to prato
```

```
python manage.py migrate
```

```
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, churras, contenttypes, pessoas, sessions
Running migrations:
  Applying churras.0002_prato_pessoa... OK
```

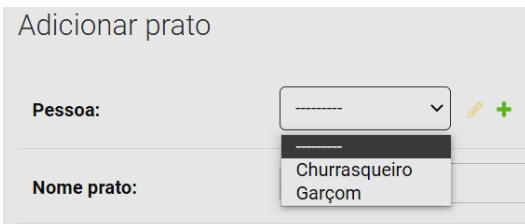
14. Acesse o admin e execute o procedimento para adicionar um prato, observe que teremos um combo para escolher a pessoas vinculada ao prato, porém não é exibido o nome, e sim os 'objects':



15. Abra 'pessoas/models.py' e corrija criando a seguinte função, dentro da classe:

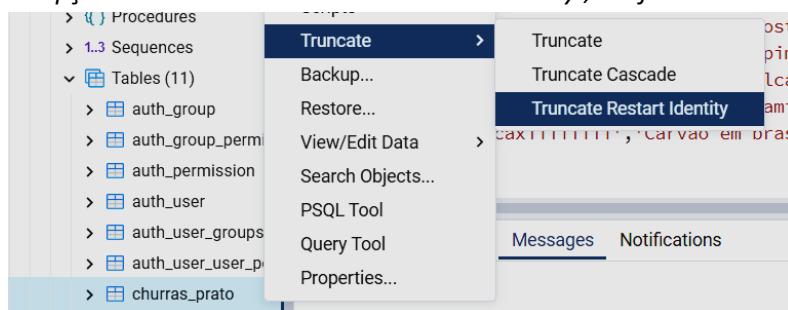
```
from django.db import models

# Create your models here.
class Pessoa(models.Model):
    nome = models.CharField(max_length=200)
    email = models.CharField(max_length=200)
    def __str__(self):
        return self.nome
```

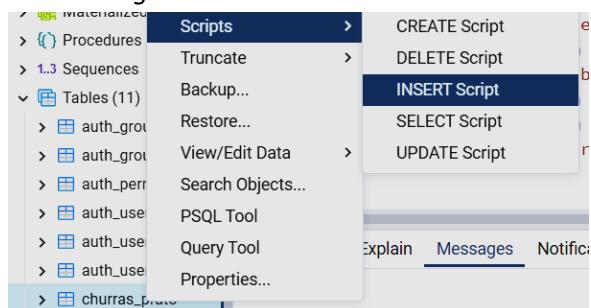


16. Cadastro alguns pratos de teste ou siga os passos a para inserir vários registros.

17. Para haver um número razoável de registros para os próximos testes vamos, copiar e executar o código sql dentro do postgres, para isso clique com o botão direito no banco 'churras', escolhas as opções 'Truncate >> Truncate Restart Identity', confirme com 'Ok'; Isso irá 'limpar' a tabela.



18. Logo após novamente com o botão direito 'Scripts >> INSERT Script', e na janela que irá abrir cole o código abaixo e execute:



```

INSERT INTO public.churras_prato(
    pessoa_id, nome_prato, ingredientes, modo_preparo,
tempo_preparo, rendimento, categoria, date_prato)
VALUES
    (1,'Picanha','Carvão em brasa, Picanha maturada e Sal
Grosso','Considerada por muitos como a mais nobre e procurada carne
de churrasco, a picanha pode ser servida ao ponto, malpassada ou bem
passada. Suculenta e com sua característica capa de
gordura',30,3,'Churrasco','2022-01-25 21:04:31-03'),
    (1,'Apfelstrudel','Massa folheada, Maçã, Canela e
Sorvete','Sobremesa tradicional austro-germânica e um delicioso
folhado de maçã e canela com sorvete',40,2,'Sobremesa','2022-01-30
21:05:50-03'),
    (2,'Picanha ao alho','Carvão em brasa, Picanha maturada, alho
e Sal Grosso','Esta é a combinação do sabor inconfundível da picanha
com o aroma acentuado do alho. Condimento que casa perfeitamente com
este corte nobre', 40,2,'Churrasco','2022-01-31 21:05:50-03'),
    (1,'Fraldinha','Carvão em brasa, Fraldinha e Sal Grosso','Uma
das carnes mais suculentas do cardápio. Requintada, com maciez
particular e pouca gordura, e uma carne que chama atenção pela sua
textura', 40,2,'Churrasco','2022-01-24 21:05:50-03'),
    (1,'Costela','Carvão em brasa, Costela e Sal Grosso','A mais
procurada! Feita na churrasqueira ou ao fogo de chão, e preparada por
mais de 08 horas para atingir o ponto ideal e torná-la a referência
de nossa churrascaria', 40,2,'Churrasco','2022-01-22 21:05:50-03'),

```

```

        (2,'Cupim', 'Carvão em brasa, Cupim e Sal Grosso', 'Uma
referência especial dos paulistas. Bastante gordurosa e macia, o
cupim é uma carne fibrosa, que se desfia quando bem-preparada ', 
40,2,'Churrasco','2022-01-29 21:05:50-03'),
        (1,'Alcatra','Carvão em brasa, Alcatra e Sal Grosso','Carne
com muitas fibras, porém macia. Sua lateral apresenta uma boa parcela
de gordura. Equilibrando de forma harmônica maciez e fibras.', 
40,2,'Churrasco','2022-01-30 21:05:50-03'),
        (2,'Maminha','Carvão em brasa, Maminha e Sal Grosso','Vem da
parte inferior da Alcatra. É uma carne com fibras, porém macia e
saborosa.', 40,2,'Churrasco','2022-01-22 21:05:50-03'),
        (2,'Abacaxiiiiii','Carvão em brasa, Abacaxi e
Canela','Abacaxi assado com canela ao creme de leite condensado ', 
40,2,'Sobremesa','2022-01-29 21:05:50-03')
;

```

19. Mostraremos o nome da pessoas a qual está vinculado o prato, então acesse, 'churras/templates/churrasco.html', e ajuste o código da seguinte maneira:

```

<h6>Preparo: {{prato.tempo_preparo}} minutos</h6>
<h6>Rendimento: {{prato.rendimento}} porções</h6>
<h6>Categoria: {{prato.categoria}}</h6>
<h6>Por: {{prato.pessoa}} </h6>

```

Picanha ao alho

```

Preparo: 40 minutos
Rendimento: 2 porções
Categoria: Churrasco
Por: Garçom

```

20. Ainda nessa atividade vamos estabelecer um controle/filtro para a publicação de pratos. Abra o arquivo 'churras/models.py', acrescente a classe mais um campo:

```

categoria = models.CharField(max_length=100)
date_prato = models.DateTimeField(default=datetime.now, blank=True)
publicado = models.BooleanField(default=False)

```

21. Execute os comandos para marcar e executar a migração:

```

python manage.py makemigrations
python manage.py migrate
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py makemigrations
Migrations for 'churras':
    churras\migrations\0003_prato_publicada.py
      - Add field publicada to prato

(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, churras, contenttypes, pessoas, sessions
Running migrations:
  Applying churras.0003_prato_publicada... OK

```

22. Acesse o admin e clique para editar um prato, ao fim do formulário verifique a possibilidade de marcar como publicada o prato:

Date prato:	Data:	30/01/2022
	Hora:	21:05:50
<input type="checkbox"/> Publicado		
Apagar		

23. Exibiremos o campo de publicado na lista de pratos e tornaremos o campo editável diretamente da lista, para isso acesse 'churras/admin.py', e altere o código:

```
# Register your models here.
class ListandoPratos(admin.ModelAdmin):
    nome_pessoa = Pessoa.nome
    list_display = ('id', 'nome_prato', 'categoria', 'tempo_preparo',
    'publicado')
    list_display_links = ('id', 'nome_prato',)
    search_fields = ('nome_prato',)
    list_filter = ('categoria', 'publicado')
    list_editable = ('publicado',)
    list_per_page = 5
```

Ação: -----					<input type="button" value="Ir"/>	0 de 5 selecionados
<input type="checkbox"/>	ID	NOME PRATO	CATEGORIA	TEMPO PREPARO	PUBLICADO	
<input type="checkbox"/>	10	Abacaxiiiiiiii	Sobremesa	40	<input checked="" type="checkbox"/>	
<input type="checkbox"/>	9	Maminha	Churrasco	40	<input checked="" type="checkbox"/>	
<input type="checkbox"/>	8	Alcatra	Churrasco	40	<input type="checkbox"/>	
<input type="checkbox"/>	7	Cupim	Churrasco	40	<input checked="" type="checkbox"/>	
<input type="checkbox"/>	6	Costela	Churrasco	40	<input type="checkbox"/>	

1 9 pratos [Mostrar tudo](#) [Salvar](#)

24. No intuito de ordenar por data, exibir apenas os pratos publicados altere 'churras/views.py', volte ao admin e marque alguns prato como publicados, e verifique na página index a exibição:

```
# Create your views here.
```

```
def index(request):
    pratos = Prato.objects.order_by('-date_prato').filter(publicado=True)
```

25. Verifique se apenas os pratos publicados estão visíveis e estão ordenadas por data decrescente;

26. Cadastre outros pratos e verifique se estão aparecendo logo no início da lista. Salve e feche todos os arquivos;

Atividade 3 – Exibindo imagens com Django

Objetivos:

- Inserir, exibir e alterar imagens na aplicação;

As imagens são uma parte essencial das aplicações, portanto, é necessário criar formas de “alimentar” o banco de dados com a finalidade de trabalhar com esse recurso. Foca na pose e na próxima atividade.

1. Acesse o arquivo ‘churras/models.py’ e adicione um campo para receber o caminho da imagem e armazenar no seu banco de dados:

```
date_prato = models.DateTimeField(default=datetime.now, blank=True)
foto_prato = models.ImageField(upload_to='fotos/%Y/%m/%d',
                                blank=True)
publicado = models.BooleanField(default=False)
```

2. Antes de realizar a migração para o banco vamos fazer ajustes na aplicação, Acesse ‘canesgril/settings.py’ para realizar os ajustes para receber no diretório os arquivos de imagens, fotos, etc.

```
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'canesgril/static')
]

# Media
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
MEDIA_URL = '/media/'
```

3. Ao tentar realizar o ‘makemigrations’, irá ocorrer um erro. Instale um dos módulos que precisamos para trabalhar com arquivos de media:

```
python manage.py makemigrations
python -m pip install Pillow
```

```
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py makemigrations
SystemCheckError: System check identified some issues:

ERRORS:
churras.Prato.foto_prato: (fields.E210) Cannot use ImageField because Pillow is not installed.
    HINT: Get Pillow at https://pypi.org/project/Pillow/ or run command "python -m pip install Pillow".

(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python -m pip install Pillow
Collecting Pillow
  Downloading Pillow-9.0.0-cp39-cp39-win_amd64.whl (3.2 MB)
    [██████████] 3.2 MB 3.3 MB/s
Installing collected packages: Pillow
Successfully installed Pillow-9.0.0
```

4. Criada a classe, informe que existem dados para serem enviados, execute o código, no terminal:

```
python manage.py makemigrations
python manage.py migrate
```

```
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py makemigrations
Migrations for 'churras':
  churras\migrations\0004_prato_foto_prato.py
    - Add field foto_prato to prato

(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, churras, contenttypes, pessoas, sessions
Running migrations:
  Applying churras.0004_prato_foto_prato... OK
```

5. Devido a instalação do Pillow é necessário fechar o terminal. Abrir um novo e reiniciar o servidor com o comando:

```
python manage.py runserver
```

6. Observe no banco de dados e na edição de algum prato que estará disponível a inserção de imagem para os pratos. Insira uma imagem representativa para o produto.



7. Agora é necessário exibir essa imagem em nossa aplicação. Indique através do arquivo 'canesgril/urls.py', onde a imagens estarão alocadas.

```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('', include('churras.urls')),
    path('admin/', admin.site.urls),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

8. Acesse 'churras/templates/index.html' e altere a página para verificar se há imagem, e exiba caso contrário continue exibindo a imagem padrão;

```
<!-- Single Best Receipe Area -->
<div class="col-12 col-sm-6 col-lg-4">
  <div class="single-best-receipe-area mb-30">
    {% if prato.foto_prato == '' %}
      
    {% else %}
      
    {% endif %}
    <div class="receipe-content">
      <a href="{% url 'churrasco' prato.id %}">
        <h5>{{ prato.nome_prato }}</h5>
        <h6>{{ prato.categoria }}</h6>
      </a>
    </div>
  </div>
</div>
```

```
</div>
{%
  endfor %}
```



9. Altere também a página 'churrasco.html' para exibir a imagem com os mesmos parâmetros da página 'index.html';

```
<div class="receipe-slider owl-carousel">
  {% if prato.foto_prato == '' %}
    
  {% else %}
    
  {% endif %}
</div>
```

10. Faça todos os testes possíveis, inclua novas imagens. Salve e feche os arquivos para finalizar a atividade.

Atividade 4 – Construindo um sistema de busca

Objetivos:

- Criar e configurar um sistema de busca e exibição de dados.

Quando se trata de dados é preciso sempre buscar informações específicas dentro de uma página, aplicação, software etc. Então os sistemas de busca e exibição de dados são fundamentais para um app, preste atenção pois esse é o nosso objetivo.

1. Ao tentar realizar uma busca é exibido o erro 'Proibido' pois ainda não foi desenvolvida essa função dentro da nossa aplicação. Criaremos uma rota para identificar a busca. Abra o arquivo 'churras/urls.py', e inclua mais um caminho:

```
urlpatterns = [
  path('', views.index, name='index'),
  path('<int:prato_id>', views.churrasco, name='churrasco'),
  path('buscar', views.buscar, name='buscar')
]
```

2. Em seguida crie um método para 'buscar', dentro do arquivo 'churras/views.py', com o seguinte código ao fim do documento:

```
return render(request, 'churrasco.html', {'prato_a_exibir'})
```

```
def buscar(request):
```

```
        return render(request, 'buscar.html')
```

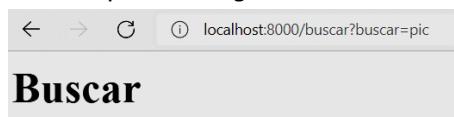
3. Crie um arquivo dentro do diretório ‘churras/templates’ chamado ‘buscar.html’, que irá receber e processar o sistema de busca. Inclua apenas uma linha de código nesse momento

```
<h1>Buscar</h1>
```

4. Abra o arquivo ‘churras/templates/index.html’ e altere a action do form para:

```
<form action="{% url 'buscar' %}">
    <input type="search" name="buscar" placeholder="O que está
    procurando...">
    <button type="submit"><i class="fa fa-search" aria-
    hidden="true"></i></button>
</form>
```

5. Realize um teste para verificar se está sendo feita a leitura e passagem do parâmetro, como no exemplo da imagem abaixo:



6. Abra a página ‘churras/templates/index.html’ e copie todo o código para o arquivo ‘churras/templates/buscar.html’. Nesse momento não será exibido nenhum resultado.

7. Acrescente ao arquivo ‘buscar.html’ uma informação caso não seja encontrado nenhum prato:

```
{% endfor %}
{% else %}
<div class="col-12 col-sm-6 col-lg-4">
    <p>Prato não encontrado!</p>
</div>
{% endif %}
```

8. Altere o arquivo ‘churras/views.py’, para que a função realize a busca e filtre os dados:

```
def buscar(request):
    pratos = Prato.objects.order_by('-date_prato').filter(publicado=True)

    if 'buscar' in request.GET:
        nome_a_buscar = request.GET['buscar']
        if nome_a_buscar:
            pratos = pratos.filter(nome_prato__icontains=nome_a_buscar)

    dados = {
        'lista_pratos' : pratos
    }

    return render(request, 'buscar.html', dados)
```

9. Crie uma ‘partials’ com o form de busca, em um novo arquivo ‘churras/templates/partials/busca.html’, recorte do código do index e cole no novo arquivo:

```
{% load static %}

<!-- Search Wrapper -->
<div class="search-wrapper">
    <!-- Close Btn -->
    <div class="close-btn"><i class="fa fa-times" aria-hidden="true"></i></div>

    <div class="container">
        <div class="row">
            <div class="col-12">
                <form action="{% url 'buscar' %}">
                    <input type="search" name="buscar" placeholder="O que está procurando..."/>
                    <button type="submit"><i class="fa fa-search" aria-hidden="true"></i></button>
                </form>
            </div>
        </div>
    </div>
</div>
```

10. No lugar do código retirado da página inclua, repita o processo de inserção do ‘partials’ nas páginas ‘buscar.html’ e ‘churrasco.html’

```
</div>

{% include 'partials/busca.html' %}

{% include 'partials/menu.html' %}
```

11. Para finalizar faça todos os testes possíveis, salve e feche os arquivos.

Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extra para rever e aprimorar o que trabalhamos até aqui:

1. Revise os códigos utilizados e verifique se há a possibilidade de utilizar mais elementos ‘partials’;

CAPÍTULO 8 – Autorizações, usuários e refatoração

Toda a aplicação essencialmente tem critérios de segurança para autorização dos usuários acessarem áreas ou ações específicas do app, então vamos nos aprofundar nesse estudo.

Durante o desenvolvimento das questões de usuários, vamos melhorar os aspectos de layout e cadastro de usuários, exibição de pratos e formulários.

Objetivos:

- Customizar aspectos de segurança dos usuários na área administrativa, e páginas de cadastro;
- Conhecer e utilizar elementos de segurança dos módulos do Django
- Implementar autenticação do usuário;
- Criar páginas com registros específicos do usuário;
- Utilizar recurso de logout.
- Criar e configurar formulários para inserção de dados, mensagens de erro e sucesso.
- Otimizar a legibilidade do código

Atividade 1 – Configurando usuários e autorizações

Objetivos:

- Customizar aspectos de segurança dos usuários na área administrativa

Customizaremos a Administração da aplicação, pois no Django, é possível criar um usuário capaz de tomar decisões e realizar modificações e outros que não possuem essas autorizações. Aproveitaremos para entender e fazer algumas correções. Veremos isso na prática nessa atividade.

1. *Acesse o admin do Django em 'http://127.0.0.1:8000/admin' e veja que até agora não trabalhamos com grupos e usuários;*

The screenshot shows the Django Admin interface. At the top, there's a header 'Administração do Site'. Below it, a blue navigation bar labeled 'AUTENTICAÇÃO E AUTORIZAÇÃO' contains two tabs: 'Grupos' and 'Usuários'. Each tab has a green '+' button for 'Adicionar' and a yellow pencil icon for 'Modificar'.

2. *Para entender melhor esse funcionamento vamos adicionar um usuário, 'Cozinheira', senha 'churras1234':*

Adicionar usuário

Primeiro, informe seu nome de usuário e senha. Então, você poderá editar outras opções do usuário.

Usuário:	<input type="text" value="Cozinheira"/>	Obrigatório. 150 caracteres ou menos. Letras, números e @/./+/-/_ apenas.
Senha:	<input type="password" value="*****"/>	Sua senha não pode ser muito parecida com o resto das suas informações pessoais. Sua senha precisa conter pelo menos 8 caracteres. Sua senha não pode ser uma senha comumente utilizada. Sua senha não pode ser inteiramente numérica.
Confirmação de senha:	<input type="password" value="churras1234"/> 	Informe a mesma senha informada anteriormente, para verificação.
<input type="button" value="Salvar e adicionar outro(a)"/> <input type="button" value="Salvar e continuar editando"/> <input type="button" value="SALVAR"/>		

3. Ao criar o usuário será redirecionado automaticamente a modificar o mesmo, e acrescentar informações e permissões e salve o usuário:

- Indicaremos como membro da equipe;
- Adicione as permissões conforme a indicação abaixo;

Permissões

<input checked="" type="checkbox"/> Ativo	Indica que o usuário será tratado como ativo. Ao invés de excluir contas de usuário, desmarque isso.
<input checked="" type="checkbox"/> Membro da equipe	Indica que usuário consegue acessar este site de administração.
<input type="checkbox"/> Status de superusuário	Indica que este usuário tem todas as permissões sem atribuí-las explicitamente.

Permissões do usuário:

permissões do usuário disponíveis 

- auth | permission | Can delete permission
- auth | permissão | Can view permission
- auth | usuário | Can add user
- auth | usuário | Can change user
- auth | usuário | Can delete user
- churras | prato | Can delete prato
- contenttypes | tipo de conteúdo | Can add content type
- contenttypes | tipo de conteúdo | Can change content type
- contenttypes | tipo de conteúdo | Can delete content type
- contenttypes | tipo de conteúdo | Can view content type
- pessoas | pessoa | Can delete pessoa
- sessions | sessão | Can add session
- sessions | sessão | Can change session

permissões do usuário escolhido(s) 

- pessoas | pessoa | Can add pessoa
- pessoas | pessoa | Can change pessoa
- pessoas | pessoa | Can view pessoa
- churras | prato | Can add prato
- churras | prato | Can change prato
- churras | prato | Can view prato

Escolher todos  **Remover todos** 

Permissões específicas para este usuário. Pressione "Control", ou "Command" no Mac, para selecionar mais de um.

4. Encerre a sessão e accese como o novo usuário ‘cozinheira’ e note que a administração mudou, não há grupos e usuários:

Administração do Django

Administração do Site

CHURRAS	 Adicionar	 Modificar
Pratos		
PESSOAS	 Adicionar	 Modificar
Pessoas		

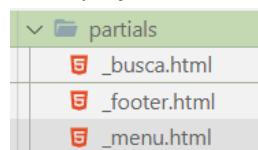
- Navegue pelos pratos e pessoas e note que é possível, adicionar e alterar os objetos, mas não é possível excluir os existentes.
- Durante o acesso a página notamos que alterando um prato ele não exibe o nome do prato e sim o objeto. Acesse 'churras/models.py' para acrescentar uma função e fazer a correção

```

publicado = models.BooleanField(default=False)
def __str__(self):
    return self.nome_prato

```

- Acesse a pasta 'churras/templates/partials' e renomeie os arquivos colocando um underline como prefixo, isso irá indicar que esse arquivo se refere a um partials onde ele estiver;



- Abra os arquivos em as partials estão associadas e faça a correção do nome do arquivo;

```

{% include 'partials/_busca.html' %}

{% include 'partials/_menu.html' %}

{% include 'partials/_footer.html' %}

```

- Faça todos os testes possível, salve e feche os arquivos utilizados.

Atividade 2 – Usuários, cadastro e segurança

Objetivos:

- Criar página customizada para cadastro do usuário
- Conhecer e utilizar elementos de segurança dos módulos do Django

Para ampliar as possibilidades de uso dos usuários, utilizaremos a própria tabela de usuários do Django e assim o usuário poderá vincular seus próprios pratos e deixá-los em uma área somente sua ou publicar de forma geral. Mão na massa ou no churras, #boratrabalhar.

- Crie um app de usuários utilizando o comando:

```
python manage.py startapp usuarios
```

- Acesse 'canesgrill/settings.py' e registre o app:

```

INSTALLED_APPS = [
    'pessoas',
    'churras',
]

```

```
'usuarios',
'django.contrib.admin',
```

3. Indicaremos em 'canesgril/urls.py' o path nas configurações gerais para os usuários

```
urlpatterns = [
    path('', include('churras.urls')),
    path('usuarios/', include('usuarios.urls')),
    path('admin/', admin.site.urls),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

4. Configure também os caminhos dos usuarios, criando um arquivo 'usuarios/urls.py' e adicionando o seguinte código:

```
from django.urls import path

from . import views

urlpatterns = [
    path('cadastro', views.cadastro, name='cadastro'),
    path('login', views.login, name='login'),
    path('dashboard', views.dashboard, name='dashboard'),
    path('logout', views.logout, name='logout'),
]
```

5. Criaremos logo em seguida os métodos para as funções estabelecidas no caminho. Acesse 'usuarios/views.py':

```
from django.shortcuts import render

# Create your views here.

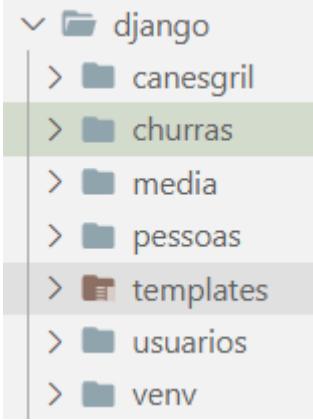
def cadastro(request):
    return render(request, 'cadastro.html')

def login(request):
    return render(request, 'login.html')

def logout(request):
    pass

def dashboard(request):
    pass
```

6. Uma vez que nossa aplicação cresceu e vamos utilizar outras páginas e templates, vamos mover a pasta templates para o diretório 'django' que é a raiz do projeto e ajustar 'canesgril/settings' para utilizar esse caminho:



```

TEMPLATES = [
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [os.path.join(BASE_DIR, 'templates')],
```

7. Acesse a aplicação e faça todos os testes possíveis;
8. Crie a página de 'templates/login.html' utilizando o código abaixo:

```

{% extends 'base.html' %}
{% load static %}
{% block content %}
<!-- Preloader -->


<i class="circle-preloader"></i>
    



{% include 'partials/_busca.html' %}

{% include 'partials/_menu.html' %}

<!-- ##### Contact Form Area Start ##### -->


<div class="container">
        <div class="row">
            <div class="col-12">
                <div class="section-heading">
                    <h2>Login</h2>
                </div>
            </div>
        </div>
        <div class="row">
            <div class="col-12">
                <div class="contact-form-area">
                    <form action="" method="">
                        <div class="row">
                            <div class="col-12 col-lg-12">
                                <label for="email"><b>Email</b></label>


```

```
<input type="text" id="email"
class="form-control" name="email" placeholder="Entre com o email
 cadastrado">
</div>
<div class="col-12 col-lg-12">
<label
for="senha"><b>Senha</b></label>
<input id="senha" type="password"
class="form-control" name="senha" placeholder="Digite aqui sua
senha">
</div>
<div class="col-12 text-center">
<button class="btn btn-success"
type="submit">Acesse sua conta</button>
<button type="button" class="btn btn-primary"
onclick="exibirCriarConta()">Faça seu Cadastro</button>
</div>
</div>
</form>
</div>
</div>
</div>
</div>

<script>
function exibirCriarConta()
{
    location.href = "{% url 'cadastro' %}";
}
</script>

{% include 'partials/_footer.html' %}
{% endblock %}
```

9. Crie a página de 'templates/cadastro.html' utilizando o código abaixo:

```
{% extends 'base.html' %}  
{% load static %}  
{% block content %}  
<!-- PreLoader -->  
<div id="preloader">  
    <i class="circle-preloader"></i>  
      
</div>  
  
{% include 'partials/_busca.html' %}  
  
{% include 'partials/_menu.html' %}
```

```

<!-- ##### Contact Form Area Start ##### -->
<div class="contact-area section-padding-0-80">
    <div class="container">
        <div class="row">
            <div class="col-12">
                <div class="section-heading">
                    <h2>Criar conta</h2>
                </div>
            </div>
        </div>
        <div class="row">
            <div class="col-12">
                <div class="contact-form-area">
                    <form action="" method="">
                        <div class="row">
                            <div class="col-12 col-lg-12">
                                <label for="nome_receita"><b>Nome
completo</b></label>
                                    <input type="text" class="form-
control" name="nome" placeholder="Ex.: Anderson Iwanezuk" required>
                                </div>
                            <div class="col-12 col-lg-12">
                                <label
for="email"><b>Email</b></label>
                                    <input type="email" class="form-
control" name="email" placeholder="Ex.: iwanezuk@churras.com"
required >
                                </div>
                            <div class="col-12 col-lg-6">
                                <label
for="password"><b>Senha</b></label>
                                    <input type="password" class="form-
control" name="password" placeholder="Digite sua senha" required>
                                </div>
                            <div class="col-12 col-lg-6">
                                <label for="password2"><b>Confirmação
de senha</b></label>
                                    <input type="password" class="form-
control" name="password2" placeholder="Digite sua senha novamente"
required>
                                </div>
                            <div class="col-12 text-center">
                                <button class="btn btn-success"
type="submit">Criar sua conta</button>
                            </div>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>

```

```

        </div>
    </div>
</div>
</div>
</div>

{% include 'partials/_footer.html' %}
{% endblock %}

```

10. Abra o arquivo ‘cadastro.html’ e inicie os procedimentos para vincular o form a tabela de usuários do banco de dados inserindo os códigos no form:

```

<div class="contact-form-area">
    <form action="{% url 'cadastro' %}" method="post">
        <div class="row">

```

11. Criaremos um método para verificar o cadastro do usuário. Acesse ‘usuarios/views.py’:

```

from django.shortcuts import render, redirect

# Create your views here.
def cadastro(request):
    if request.method == 'POST':
        print('Usuário criado com sucesso')
        return redirect('login')
    else:
        return render(request, 'cadastro.html')

```

12. Realize um teste, porém uma mensagem de erro será exibida;

- a. CSRF ou cross-site request forgery, que significa “falsificação de solicitação entre sites”, ou “ataque de um clique”. Sendo assim, o Django oferece proteção contra esse tipo de ataque.

Proibido (403)

Verificação CSRF falhou. Pedido cancelado.

13. Para correção, em ‘cadastro.html’ insira o código na primeira linha do form e ao abrir o formulário um código de segurança será gerado;

```

<form action="{% url 'cadastro' %}" method="post">
    {% csrf_token %}
    <div class="row">

```

```

▼ <form action="/usuarios/cadastro" method="post">
    <input type="hidden" name="csrfmiddlewaretoken" value="lIxUUvUp
    VXp9iN19YI01H2XG1tKDvsRAeK18CBCqJrhTK64dAn75ampETiYOKKAi">
    <div class="row">

```

14. Execute o cadastro novamente e envie, você será redirecionado para página de login, e no terminal haverá um indicativo de sucesso da inserção;

```

[02/Feb/2022 00:46:31] "GET /usuarios/cadastro HTTP/1.1" 200 8441
Usuário criado com sucesso

```

15. Revisitamos ‘usuarios/views.py’, para criar algumas validações, e enviar os dados para tabela ‘auth_user’

```
from django.shortcuts import render, redirect
from django.contrib.auth.models import User

# Create your views here.
def cadastro(request):
    if request.method == 'POST':
        nome = request.POST['nome']
        email = request.POST['email']
        senha = request.POST['password']
        senha2 = request.POST['password2']
        if not nome.strip():
            print('O campo nome não pode ficar em branco')
            return redirect('cadastro')
        if not email.strip():
            print('O campo email não pode ficar em branco')
            return redirect('cadastro')
        if senha != senha2:
            print('As senhas não são iguais')
            return redirect('cadastro')
        if User.objects.filter(email=email).exists():
            print('Usuário já cadastrado')
            return redirect('cadastro')
        user = User.objects.create_user(username=nome, email=email,
password=senha)
        user.save()
        print('Usuário cadastrado com sucesso')
        return redirect('login')
    else:
        return render(request, 'cadastro.html')
```

16. Realize vários testes da validação do formulário, observando pelo terminal as mensagens;

```
O campo nome não pode ficar em branco
[02/Feb/2022 01:13:14] "POST /usuarios/cadastro HTTP/1.1" 302 0
[02/Feb/2022 01:13:14] "GET /usuarios/cadastro HTTP/1.1" 200 8441
As senhas não são iguais
[02/Feb/2022 01:14:07] "POST /usuarios/cadastro HTTP/1.1" 302 0
[02/Feb/2022 01:14:07] "GET /usuarios/cadastro HTTP/1.1" 200 8441
Usuário já cadastrado
[02/Feb/2022 01:16:08] "POST /usuarios/cadastro HTTP/1.1" 302 0
[02/Feb/2022 01:16:08] "GET /usuarios/cadastro HTTP/1.1" 200 8441
```

17. Crie uma conta para testar, e observe no terminal e no banco se os dados foram inseridos;

Criar conta

Nome completo	
caixa	
Email	
caixa@churras.com	
Senha	Confirmação de senha
....	1234
Criar sua conta	

Usuário cadastrado com sucesso
[02/Feb/2022 01:08:25] "POST /usuarios/cadastro HTTP/1.1" 302 0
[02/Feb/2022 01:08:25] "GET /usuarios/login HTTP/1.1" 200 7863

username	first_name	last_name	email	is_staff	is_active	date_joined
boi			boi@churras.com	true	true	2022-01-30 21:04:06.010995-03
Cozinheira				true	true	2022-02-01 22:23:02-03
caixa			caixa@churras.com	false	true	2022-02-02 01:08:24.803123-03

18. Finalizamos a atividade, salvando e fechando os arquivos utilizados.

Atividade 3 – Login, Dashboard e logout

Objetivos:

- Implementar autenticação do usuário;
- Criar página com os pratos criados pelo usuário;
- Utilizar recurso de logout.

Para acessar determinadas áreas da aplicação e ter seus dados restritos à sua visualização é preciso um sistema de autenticação funcional e um dashboard capaz de mostrar as informações relativas ao seu usuário. Esse é o nosso objetivo.

1. Abra o arquivo 'login.html' e vincule o form as ações de login e a segurança CSRF:

```
<div class="contact-form-area">
    <form action="{% url 'login' %}" method="post">
        {% csrf_token %}
        <div class="row">
```

2. Crie uma página 'templates/dashboard.html' apenas para evitar erros no código, logo utilizaremos esse recurso. Acrescente apenas uma linha ao arquivo.

```
<h1>DashBoard</h1>
```

3. Abra 'usuarios/views.py', para criar algumas validações, e enviar os dados para tabela 'auth_user'

```
def login(request):
    if request.method == 'POST':
```

```

        email = request.POST['email']
        senha = request.POST['senha']
        if email == "" or senha == "":
            print('Os campos email e senha não podem ficar em
branco')
            return redirect('login')
        print(email, senha)
        return redirect('dashboard')
    return render(request, 'login.html')

def logout(request):
    pass

def dashboard(request):
    return render(request, 'dashboard.html')

```

4. Realize todos os testes possíveis observando os resultados no terminal;
5. Voltamos ao 'views.py' para fazer com que a aplicação compare os dados com o banco e faça a autenticação do usuário; importe o módulo 'auth' e siga atentamente os códigos;
 - a. Para saber mais sobre autenticação no Django, ele tem um documentação muita completa, acesse <https://docs.djangoproject.com/en/4.0/topics/auth/>

```

from django.contrib import auth

def login(request):
    if request.method == 'POST':
        email = request.POST['email']
        senha = request.POST['senha']
        if email == "" or senha == "":
            print('Os campos email e senha não podem ficar em
branco')
            return redirect('login')
        print(email, senha)
        if User.objects.filter(email=email).exists():
            nome = User.objects.filter(email=email).values_list(
                'username', flat=True).get()
            user = auth.authenticate(request, username=nome,
password=senha)
            if user is not None:
                auth.login(request, user)
                print('Login realizado com sucesso')
                return redirect('dashboard')
    return render(request, 'login.html')

```

6. Faça os testes de autenticação e logo após insira um login válido. Verifique pelo terminal se o login obteve sucesso.

```

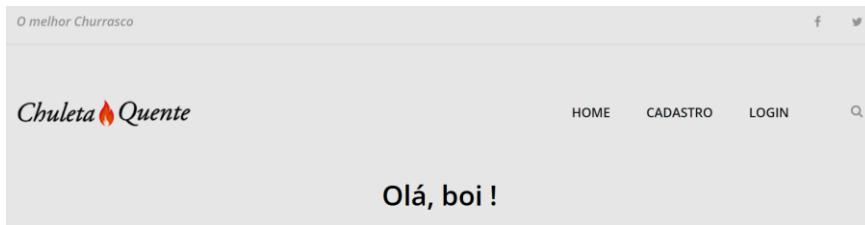
Login realizado com sucesso
[02/Feb/2022 02:08:28] "POST /usuarios/login HTTP/1.1" 302 0
[02/Feb/2022 02:08:28] "GET /usuarios/dashboard HTTP/1.1" 200 18

```

7. Copie todo o código da página ‘index.html’ abaixo e cole dentro da página ‘dashboard.html’, e acrescente o trecho abaixo logo abaixo do partials do menu. Observe o resultado;

```
{% include 'partials/_menu.html' %}

<div class="contact-area section-padding-0-5">
    <div class="container">
        <div class="row">
            <div class="col-12">
                <div class="section-heading">
                    <h3>Olá, {{ user.username }} !</h3>
                </div>
            </div>
        </div>
    </div>
</div>
```



8. Verifique se o usuário está logado e exiba os links de acordo com a condição, então abra ‘_menu.html’ e insira os códigos;

```
<ul>
    {% if user.is_authenticated %}
        <li><a href="{% url 'index' %}">Página principal</a></li>
        <li><a href="{% url 'dashboard' %}">Meus Pratos</a></li>
        <li><a href="{% url 'logout' %}">Logout</a></li>
    {% else %}
        <li><a href="{% url 'index' %}">Home</a></li>
        <li><a href="{% url 'cadastro' %}">Cadastro</a></li>
        <li><a href="{% url 'login' %}">Login</a></li>
    {% endif %}
</ul>
```

9. Abra ‘views.py’ para inserir o código para a função logout e restringir a visualização do dashboard apenas quando o usuário estiver logado;

```
def logout(request):
    auth.logout(request)
    return redirect('index')

def dashboard(request):
    if request.user.is_authenticated:
        return render(request, 'dashboard.html')
```

```
    else:  
        return redirect('index')
```

10. Teste novamente o login e logout e o acesso direto ao dashboard;
11. Salve e feche os arquivos para finalizar a atividade.

Atividade 4 – Formulários e mensagens

Objetivos:

- Criar e configurar formulários para inserção de dados.
- Criar e configurar mensagens de erro e sucesso ligadas as ações do formulário.
- Otimizar a legibilidade do código

Todas as informações inseridas numa aplicação utilizam formulários para isso, então daremos uma atenção especial a esse elemento importante do Django. Desafio aceito.

1. Criaremos um path para a página ‘cria_prato’. Acesse ‘usuarios/urls.py’ e insira o novo caminho;

```
from django.urls import path  
  
from . import views  
  
urlpatterns = [  
    path('cadastro', views.cadastro, name='cadastro'),  
    path('login', views.login, name='login'),  
    path('dashboard', views.dashboard, name='dashboard'),  
    path('logout', views.logout, name='logout'),  
    path('cria/prato', views.cria_prato, name='cria_prato'),  
]
```

2. Abra ‘usuarios/views.py’ para inserir o código para a função cria_prato, adicione ao fim do documento os códigos;

```
def cria_prato(request):  
    return render(request, 'cria_prato.html')
```

3. Abra ‘_menu.html’ e insira um link para página que cria o prato;

```
<ul>  
    {% if user.is_authenticated %}  
        <li><a href="{% url 'index' %}">Página principal</a></li>  
        <li><a href="{% url 'dashboard' %}">Meus Pratos</a></li>  
        <li><a href="{% url 'cria_prato' %}">Criar Pratos</a></li>  
        <li><a href="{% url 'logout' %}">Logout</a></li>  
    {% else %}  
        <li><a href="{% url 'index' %}">Home</a></li>  
        <li><a href="{% url 'cadastro' %}">Cadastro</a></li>  
        <li><a href="{% url 'login' %}">Login</a></li>
```

```
{% endif %}  
</ul>
```

4. Em seguida utilize o código abaixo para criar o arquivo 'templates/cria_prato.html'

```
{% extends 'base.html' %}  
{% load static %}  
{% block content %}  
<!-- Preloader -->  
<div id="preloader">  
    <i class="circle-preloader"></i>  
      
</div>  
  
{% include 'partials/_busca.html' %}  
  
{% include 'partials/_menu.html' %}  
  
<!-- ##### Contact Form Area Start ##### -->  
<div class="contact-area section-padding-0-80">  
    <div class="container">  
        <div class="row">  
            <div class="col-12">  
                <div class="section-heading">  
                    <h3>Crie seu prato, {{ user.username }} :)</h3>  
                </div>  
            </div>  
        </div>  
  
        <div class="row">  
            <div class="col-12">  
                <p>Todos os campos são obrigatórios</p>  
                <div class="contact-form-area">  
                    <form action="{% url 'cria_prato' %}"  
method="post" enctype="multipart/form-data">  
                        {% csrf_token %}  
                        <div class="row">  
                            <div class="col-12 col-lg-12">  
                                <label for="nome_prato"><b>Título do  
prato</b></label>  
                                <input type="text" id="nome_prato"  
class="form-control" name="nome_prato" placeholder="Ex. Maminha no  
limão" required>  
                            </div>  
                            <div class="col-12">
```

```

        <label
for="file"><b>Ingredientes</b></label>
        <textarea class="form-control"
name="ingredientes" cols="30" rows="10" placeholder="Ex. Maminha,
&#10; 3 Limões, &#10;Sal e temperos" required></textarea>
        </div>
        <div class="col-12">
            <label for="file"><b>Modo de
preparo</b></label>
            <textarea class="form-control"
name="modo_preparo" cols="30" rows="10" placeholder="Ex. Corte
saboroso, temperado com lima, sal e diversos temperos, marinando para
obter o máximo de suculência." required></textarea>
        </div>
        <div class="col-12 col-lg-3">
            <label for="file"><b>Tempo de preparo
(minutos)</b></label>
            <input type="number" class="form-
control" name="tempo_preparo" placeholder="Ex. 2" required>
        </div>
        <div class="col-12 col-lg-3">
            <label
for="file"><b>Rendimento</b></label>
            <input type="text" class="form-
control" name="rendimento" placeholder="Ex. serve 1 pessoa" required>
        </div>
        <div class="col-12 col-lg-3">
            <label for="file"><b>Categoria da
prato</b></label>
            <input type="text" class="form-
control" name="categoria" placeholder="Ex. Churrasco" required>
        </div>
        <div class="col-12 col-lg-3">
            <label for="file"><b>Foto</b></label>
            <input type="file" class="form-
control" name="foto_prato" required>
        </div>
        <div class="col-12 text-center">
            <button class="btn delicious-btn mt-
30" type="submit">Criar sua prato</button>
        </div>
    </div>
</form>
</div>
</div>
</div>
</div>

```

```
{% include 'partials/_footer.html' %}  
{% endblock %}
```

Crie seu prato, boi :)

Todos os campos são obrigatórios

Título do prato
Ex. Maminha no limão

Ingredientes
*Ex. Maminha,
3 Limões,
Sal e temperos*

Modo de preparo
Ex. Corte saboroso, temperado com limão, sal e diversas temperos, marinando para obter o máximo de suculência.

Tempo de preparo (minutos) <i>Ex. 2</i>	Rendimento <i>Ex. serve 1 pessoa</i>	Categoria da prato <i>Ex. Churrasco</i>	Foto <input type="button" value="Escolher Arquivo"/> Nenh... carregado
--	---	--	---

Criar Seu Prato

5. Vamos alterar o relacionamento do prato, da pessoa para o usuário. Abra o arquivo 'churras/models.py' e edite o início do arquivo como abaixo;

```
from django.db import models
from datetime import datetime
# from pessoas.models import Pessoa
from django.contrib.auth.models import User

# Create your models here.
class Prato(models.Model):
    # pessoa = models.ForeignKey(Pessoa, on_delete=models.CASCADE)
    pessoa = models.ForeignKey(User, on_delete=models.CASCADE)
    nome_prato = models.CharField(max_length=100)
```

6. Para atualizar a base de dados, execute os comandos para marcar e executar a migração, após esse procedimento o prato estará vinculado ao usuário;

```
python manage.py makemigrations
python manage.py migrate
```

```
(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py makemigrations
Migrations for 'churras':
  churras\migrations\0005_alter_prato_pessoa.py
    - Alter field pessoa on prato

(venv) C:\Users\iwane\OneDrive - sp.senac.br\GD\Python II\ativ\django>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, churras, contenttypes, pessoas, sessions
Running migrations:
  Applying churras.0005_alter_prato_pessoa... OK
```

7. Abra 'usuarios/views.py' para inserir o que envia os dados para a base de dados, utilizando a função `cria_prato`:

```
from django.shortcuts import get_object_or_404, redirect, render
from django.contrib.auth.models import User
from django.contrib import auth
from churras.models import Prato

def cria_prato(request):
    if request.method == 'POST':
        nome_prato = request.POST['nome_prato']
        ingredientes = request.POST['ingredientes']
        modo_preparo = request.POST['modo_preparo']
        tempo_preparo = request.POST['tempo_preparo']
        rendimento = request.POST['rendimento']
        categoria = request.POST['categoria']
        foto_prato = request.FILES['foto_prato']
        user = get_object_or_404(User, pk=request.user.id)
        prato = Prato.objects.create(pessoa=user,
                                      nome_prato=nome_prato, ingredientes=ingredientes,
                                      modo_preparo=modo_preparo, tempo_preparo=tempo_preparo,
                                      rendimento=rendimento, categoria=categoria, foto_prato=foto_prato)
        prato.save()
        return redirect('dashboard')
    else:
        return render(request, 'cria_prato.html')
```

8. Acrescente também a função `dashboard`, os dados abaixo, assim serão exibidos os pratos cadastrados por um usuário específico;

```
def dashboard(request):
    if request.user.is_authenticated:
        id = request.user.id
        pratos = Prato.objects.order_by('-date_prato').filter(
            pessoa=id)

        dados = {
            'lista_pratos' : pratos
        }
        return render(request, 'dashboard.html', dados)
    else:
        return redirect('index')
```

9. Cadastre uma receita para realizar os testes;

10. Para exibir mensagens de erros é possível utilizar alguns dos recursos do Django. Abra 'canesgril/settings.py' e ao fim do documento vamos informar configurações das mensagens;

a. Saiba mais acessando: <https://docs.djangoproject.com/en/4.0/ref/contrib/messages/>

```
# Messages
from django.contrib.messages import constants as messages
```

```

MESSAGE_TAGS = {
    messages.ERROR: 'danger',
    messages.SUCCESS: 'success',
}

```

11. Crie um arquivos 'partials/_alertas.html' para armazenar as mensagens, conforme o código a seguir:

```

{% if messages %}
    {% for message in messages %}
        <div class="alert alert-{% message.tags %}" role="alert">
            {{ message }}
        </div>
    {% endfor %}
{% endif %}

```

12. Inclua no arquivo 'login.html' e 'cadastro.html' a partial recém-criada pouco antes do form, conforme abaixo:

```

<h2>Login</h2>
</div>
</div>
</div>
{% include 'partials/_alertas.html' %}
<div class="row">
    <div class="col-12">
        <div class="contact-form-area">
            <form action="{% url 'login' %}" method="post">

```

13. Abra 'usuarios/views.py' para inserir na função a exibição das mensagens;

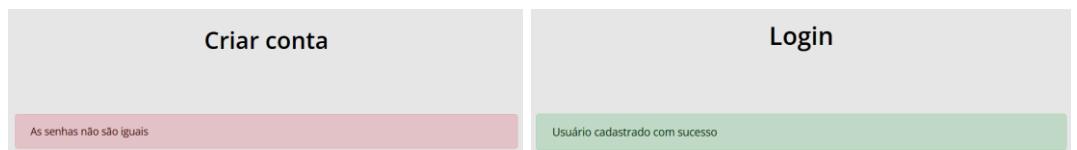
```
from django.contrib import auth, messages
```

```

def cadastro(request):
    if request.method == 'POST':
        nome = request.POST['nome']
    ...
    if senha != senha2:
        messages.error(request, 'As senhas não são iguais')
        print('As senhas não são iguais')
    ...
        user.save()
        print('Usuário cadastrado com sucesso')
        messages.success(request, 'Usuário cadastrado com sucesso')
        return redirect('login')
    else:
        return render(request, 'cadastro.html')

```

14. Realize os testes para visualizar as mensagens configuradas;



15. Vamos melhorar a legibilidade de nosso código. Abra 'usuarios/views.py' para fazermos modificações e criar funções a seguir;

```

def cadastro(request):
    if request.method == 'POST':
        nome = request.POST['nome']
        email = request.POST['email']
        senha = request.POST['password']
        senha2 = request.POST['password2']
        if campo_vazio(nome):
            messages.error(request, 'O campo nome não pode ficar em branco')
            return redirect('cadastro')
        if campo_vazio(email):
            messages.error(request, 'O campo email não pode ficar em branco')
            return redirect('cadastro')
        if senhas_nao_sao_iguais(senha, senha2):
            messages.error(request, 'As senhas não são iguais')
            return redirect('cadastro')
        if User.objects.filter(email=email).exists():
            messages.error(request, 'Usuário já cadastrado')
            return redirect('cadastro')
        if User.objects.filter(username=nome).exists():
            messages.error(request, 'Usuário já cadastrado')
            return redirect('cadastro')
        user = User.objects.create_user(username=nome, email=email, password=senha)
        user.save()
        messages.success(request, 'Usuário cadastrado com sucesso')
        return redirect('login')
    else:
        return render(request, 'cadastro.html')

def campo_vazio(campo):
    return not campo.strip()

def senhas_nao_sao_iguais(senha, senha2):
    return senha != senha2

def login(request):
    if request.method == 'POST':
        email = request.POST['email']
        senha = request.POST['senha']

```

```

if campo_vazio(email) or campo_vazio(senha):
    messages.error(request,'Os campos email e senha não podem
ficar em branco')
    return redirect('login')
if User.objects.filter(email=email).exists():
    nome = User.objects.filter(email=email).values_list(
        'username', flat=True).get()
    user = auth.authenticate(request, username=nome,
password=senha)
    if user is not None:
        auth.login(request, user)
        messages.success('Login realizado com sucesso')
        return redirect('dashboard')
return render(request, 'login.html')

```

16. Para finalizar faça todos os testes possíveis, salve e feche os arquivos.

Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extra para rever e aprimorar o que trabalhamos até aqui:

1. *Pesquise novas formas de criar e customizar o sistema de autenticação;*
2. *Crie outros exemplos de mensagens de orientação aos usuários;*

CAPÍTULO 9 – Finalizando CRUD e paginação

Na reta final de nossas atividades vamos construir elementos para apagar registros e apps, editar dados e colocar numeração nas páginas web.

Objetivos:

- Apagar registros da aplicação;
- Desvincular APPs do projeto
- Criar uma página para utilizar elementos de manipulação de dados;
- Construir um sistema de paginação
- Criar um menu dinâmico com base nas permissões do usuário

Atividade 1 – Apagando pratos e apps

Objetivos:

- Apagar registros da aplicação;
- Desvincular APPs do projeto

Dentro da nossa aplicação é preciso apagar alguns dados e até apps, isso é de grande importância, por isso esse é o nosso próximo tema.

17. Vamos cadastrar o caminho para apagar o prato. Acesse ‘usuarios/urls.py’ e insira a rota de acesso, indicando o parâmetro do id, para apagar um prato em específico;

```
path('cria/prato', views.cria_prato, name='cria_prato'),
path('deleta/<int:prato_id>', views.deleta_prato,
name='deleta_prato')
]
```

18. Criaremos a função que deleta o prato, para isso abra o arquivo ‘usuarios/views.py’ e utilize o código:

```
def deleta_prato(request, prato_id):
    prato = get_object_or_404(Prato, pk=prato_id)
    prato.delete()
    return redirect('dashboard')
```

19. Na página ‘dashboard.html’ é preciso criar um botão para executar a ação de apagar o prato, localize as informações do prato e insira um link da seguinte forma:

```
<a href="{% url 'churrasco' prato.id %}">
    <h5>{{ prato.nome_prato }}</h5>
    <h6>{{ prato.categoria }}</h6>
</a>
<a href="{% url 'deleta_prato' prato.id %}" type="button" class="btn btn-danger">
```

Apagar

20. Crie um prato, teste e logo em seguida clique no botão ‘Apagar’ para verificar o seu funcionamento;



teste
Churrasco
Apagar

21. Nesse momento já que falamos em apagar itens, vamos aproveitar para apagar o app de pessoas, pois integraremos o acesso aos usuários do Django admin. Abra o arquivo ‘canesgril/settings.py’, e apague a linha de ‘pessoas’ dos apps instalados;

```
INSTALLED_APPS = [  
    # 'pessoas',  
    'churras',  
    'usuarios',
```

22. Ao acessar novamente a aplicação ocorrerá um erro, pois algumas migrações estão registradas para o app de pessoas. Altere o arquivo ‘churras/migrations/0002_prato_pessoa’ e copie o código que referenciar novamente;

```
# Generated by Django 4.0.1 on 2022-01-31 04:46  
  
from django.conf import settings  
from django.db import migrations, models  
...  
    dependencies = [  
        # ('pessoas', '0001_initial'),  
        ('churras', '0001_initial'),  
    ]  
...  
        name='pessoa',  
        # field=models.ForeignKey(default='',  
on_delete=django.db.models.deletion.CASCADE, to='pessoas.pessoa'),  
        # preserve_default=False,  
        field=models.ForeignKey(on_delete=django.db.models.deleti  
on.CASCADE, to=settings.AUTH_USER_MODEL),  
    ),  
]
```

23. É necessário desvincular o admin ao módulo de pessoas. Altere ‘churras/admin.py’ para não buscas mais essa classe:

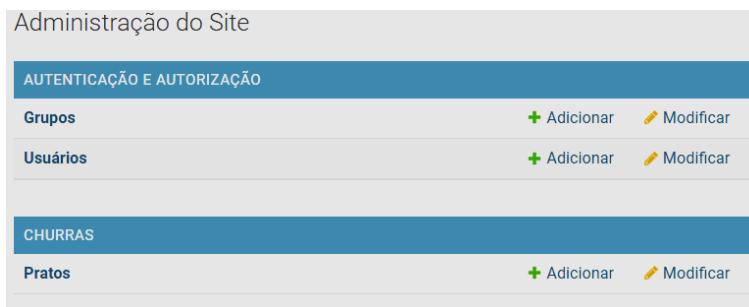
```
from django.contrib import admin
```

```

from .models import Prato
# from pessoas.models import Pessoa

# Register your models here.
class ListandoPratos(admin.ModelAdmin):
    # nome_pessoa = Pessoa.nome
    list_display =
('id', 'nome_prato', 'categoria', 'tempo_preparo', 'publicado')

```



24. Faça todos os testes possível, salve e feche os arquivos utilizados.

Atividade 2 – Edição de dados

Objetivos:

- Criar uma página e utilizar elementos para manipular dados;

Quando queremos modificar alguma informação em nossa aplicação, temos que selecionar dos dados e inserir automaticamente num formulário de edição. Após as alterações é necessário enviar os dados atualizados novamente ao banco. Esse é o desafio dessa atividade.

25. Crie um botão para modificar o prato em 'dashboard.html', adicionando antes do botar de apagar o seguinte código;

```

<a href="{% url 'edita_prato' prato.id %}" type="button" class="btn btn-info">
    Editar
</a>
<a href="{% url 'deleta_prato' prato.id %}" type="button" class="btn btn-danger">
    Apagar
</a>

```

26. Acesse 'usuarios/urls.py' e insira a rota de acesso, indicando o parâmetro do id, para alterar um prato em específico;

```

path('deleta/<int:prato_id>', views.deleta_prato,
name='deleta_prato'),

```

```
    path('edita/<int:prato_id>', views.edita_prato,
        name='edita_prato')
```

27. Criaremos a função que edita o prato, para isso abra o arquivo ‘usuarios/views.py’ e utilize o código:
28. Criaremos a função que deleta o prato, para isso abra o arquivo ‘usuarios/views.py’ e utilize o código:

```
def edita_prato(request, prato_id):
    prato = get_object_or_404(Prato, pk=prato_id)
    prato_a_editar = {'prato':prato}
    return render(request,'edita_prato.html', prato_a_editar)
```

29. Crie um arquivo na pasta templates com o nome ‘edita_prato.html’. Em seguida insira na página de edição o código disponível abaixo:

```
{% extends 'base.html' %}
{% load static %}
{% block content %}
<!-- Preloader -->


<i class="circle-preloader"></i>
    



{% include 'partials/_busca.html' %}

{% include 'partials/_menu.html' %}

<!-- ##### Contact Form Area Start ##### -->


<div class="container">
        <div class="row">
            <div class="col-12">
                <div class="section-heading">
                    <h3>Edite aqui seu prato, {{ user.username }} :)</h3>
                </div>
            </div>
        </div>
    </div>
    {% include 'partials/_alertas.html'%}
    <div class="row">
        <div class="col-12">
            <p>Todos os campos são obrigatórios</p>
            <div class="contact-form-area">
                <form action="" method="post"
enctype="multipart/form-data">
                    {% csrf_token %}


```

```

        <input type="text" id="prato_id"
class="form-control" name="prato_id" value="{{prato.id}} hidden>
        <div class="row">
            <div class="col-12 col-lg-12">
                <label for="nome_prato"><b>Título do
prato</b></label>
                    <input type="text" id="nome_prato"
class="form-control" name="nome_prato" placeholder="Ex. Maminha no
limão" value="{{prato.nome_prato}}" required>
                </div>
                <div class="col-12">
                    <label
for="file"><b>Ingredientes</b></label>
                    <textarea class="form-control"
name="ingredientes" cols="30" rows="10" placeholder="Ex. Maminha,
&#10; 3 Limões, &#10;Sal e temperos"
required>{{prato.ingredientes}}</textarea>
                </div>
                <div class="col-12">
                    <label for="file"><b>Modo de
preparo</b></label>
                    <textarea class="form-control"
name="modo_preparo" cols="30" rows="10" placeholder="Ex. Ex. Corte
saboroso, temperado com limão, sal e diversos temperos, marinando
para obter o máximo de suculência."
required>{{prato.modo_preparo}}</textarea>
                </div>
                <div class="col-12 col-lg-4">
                    <label for="file"><b>Tempo de preparo
(minutos)</b></label>
                    <input type="number" class="form-
control" name="tempo_preparo" placeholder="Ex. 2"
value="{{prato.tempo_preparo}}" required>
                </div>
                <div class="col-12 col-lg-4">
                    <label
for="file"><b>Rendimento</b></label>
                    <input type="text" class="form-
control" name="rendimento" placeholder="Ex. serve 1 pessoa"
value="{{prato.rendimento}}" required>
                </div>
                <div class="col-12 col-lg-4">
                    <label for="file"><b>Categoria do
prato</b></label>
                    <input type="text" class="form-
control" name="categoria" placeholder="Ex. Churrasco"
value="{{prato.categoria}}>required>
                </div>

```

```

        <div class="col-12 col-lg-4">
            <label for="file"><b>Foto
utilizada</b></label>
                
            </div>
        <div class="col-12 col-lg-4">
            <label for="file"><b>Adicionar nova
foto</b></label>
                <input type="file" class="form-
control" name="foto_prato">
            </div>
        <div class="col-12 text-center">
            <button class="btn btn-success"
type="submit">Atualizar</button>
        </div>
    </div>
</div>
</div>
</div>
</div>
</div>

{% include 'partials/_footer.html' %}
{% endblock %}

```

30. Execute a página ‘dashboard.html’ para visualizar o resultado até agora:



31. Clique em editar e verifique o funcionamento do formulário:

Edite aqui seu prato, boi :)

Todos os campos são obrigatórios

Título do prato	teste
Ingredientes	teste
Modo de preparo	teste

32. O formulário exibe os dados, mas não atualiza no banco, é preciso definir um action do form 'edita_prato.html'. Faça conforme a indicação:

```
<form action="{% url 'atualiza_prato %}" method="post"
enctype="multipart/form-data">
```

33. Na sequência é preciso registrar a url no arquivo 'usuarios/urls.py';

```
path('deleta/<int:prato_id>', views.deleta_prato,
name='deleta_prato'),
path('edita/<int:prato_id>', views.edita_prato,
name='edita_prato'),
path('atualiza_prato', views.atualiza_prato,
name='atualiza_prato'),
```

34. Abra 'views.py' para inserir o código para a função atualiza_prato;

```
def atualiza_prato(request):
    if request.method == 'POST':
        prato_id = request.POST['prato_id']
        p = Prato.objects.get(pk=prato_id)
        p.nome_prato = request.POST['nome_prato']
        p.ingredientes = request.POST['ingredientes']
        p.modo_preparo = request.POST['modo_preparo']
        p.tempo_preparo = request.POST['tempo_preparo']
        p.rendimento = request.POST['rendimento']
        p.categoria = request.POST['categoria']
        if 'foto_prato' in request.FILES:
            p.foto_prato = request.FILES['foto_prato']
        p.save()
    return redirect('dashboard')
```

35. Edite o nome e foto do prato, teste para verificação;



36. Finalizamos a atividade, salvando e fechando os arquivos utilizados.

Atividade 3 – Paginação

Objetivos:

- Construir um sistema de paginação

Ao trabalharmos na aplicação a tendência é que muitos dados sejam inseridos, criando listas e páginas enormes, com muitas informações. Para dividir os registros utilizamos os recursos de paginação estabelecendo quantos itens cada página vai exibir. Essa atividade vai ser bem interessante. Vamos ao desenvolvimento?

37. Abra o arquivo ‘churras/view.py’ para importarmos o paginator:

```
from django.core.paginator import Paginator, EmptyPage,
PageNotAnInteger

# Create your views here.
def index(request):
    pratos = Prato.objects.order_by('-
date_prato').filter(publicado=True)
    paginator = Paginator(pratos, 2)
    page = request.GET.get('page')
    pratos_por_pagina = paginator.get_page(page)
    dados = {
        'lista_pratos' : pratos_por_pagina
    }
```

38. Crie um arquivo ‘partials/_paginator.html’ e insira o código abaixo:

```
<!-- ##### Pagination ##### -->
<section class="top-catagory-area section-padding-20-0">
    <div class="container">
        <ul class="pagination">
            <li class="page-item">
                <a href="" class="page-link">&laquo;Anterior</a>
            </li>
            <li class="page-item disabled">
                <a class="page-link">&laquo;</a>
            </li>
```

```

        <li class="page-item active">
            <a class="page-link"></a>
        </li>
        <li class="page-item">
            <a href="" class="page-link"></a>
        </li>
        <li class="page-item">
            <a href="" class="page-link">&raquo;Próxima</a>
        </li>
        <li class="page-item disabled">
            <a class="page-link">&raquo;</a>
        </li>
    </ul>
</div>
</section>
<!-- ##### Pagination End ##### -->
```

39. Abra o arquivo 'index.html' e faça a vinculação da partial:

```

<!-- ##### Best Receipe Area End ##### -->

{% include 'partials/_paginator.html' %}

{% include 'partials/_footer.html' %}
```

40. No arquivo 'partials/_paginator.html' vamos construir vários elementos de código, a primeira condição é para exibir a paginação somente se houver pratos para exibir em outras páginas

```

<!-- ##### Pagination ##### -->
<section class="top-catalog-area section-padding-20-0">
    <div class="container">
        {% if lista_pratos.has_other_pages %}
            <ul class="pagination">
...
            </ul>
        {% endif %}
    </div>
</section>
<!-- ##### Pagination End ##### -->
```

41. Continuando a paginação, verificamos se há pratos para exibir anteriormente, caso contrário não exibiremos o link 'Anterior';

```

<!-- ##### Pagination ##### -->
...
<ul class="pagination">
    {% if lista_pratos.has_previous %}
        <li class="page-item">
            <a href="?page={{ lista_pratos.previous_page_number }}>&laquo;Anterior</a>
```

```

        </li>
        {% else %}
            <li class="page-item disabled">
                <a class="page-link">&laquo;</a>
            </li>
        {% endif %}

        ...
<!-- ##### Pagination End ##### -->
```

42. Nessa etapa vamos verificar quantas páginas temos, exibir o número, e desativar o link da página que está sendo exibida:

```

<!-- ##### Pagination ##### -->
<section class="top-catagory-area section-padding-20-0">
    ...
        </li>
    {% endif %}
    {% for pagina in lista_pratos.paginator.page_range %}
        {% if lista_pratos.number == pagina %}
            <li class="page-item active">
                <a class="page-link">{{ pagina }}</a>
            </li>
        {% else %}
            <li class="page-item">
                <a href="?page={{pagina}}" class="page-link">{{pagina}}</a>
            </li>
        {% endif %}
    {% endfor %}
    ...
</section>
<!-- ##### Pagination End ##### -->
```

43. Continuando a paginação, verificamos se há pratos para exibir posteriormente, caso contrário não exibiremos o link ‘Anterior’;

```

<!-- ##### Pagination ##### -->
<section class="top-catagory-area section-padding-20-0">
    ...
        {% endfor %}
        {% if lista_pratos.has_next %}
            <li class="page-item">
                <a href="?page={{ lista_pratos.next_page_number }}" class="page-link">&raquo;Próxima</a>
            </li>
        {% else %}
            <li class="page-item disabled">
                <a class="page-link">&raquo;</a>
            </li>
```

```

        {% endif %}
    </ul>
...
</section>
<!-- ##### Pagination End ##### -->

```

44. Salve os arquivos e navegue pela paginação;

« 1 2 3 »Próxima

«Anterior 1 2 3 »Próxima

«Anterior 1 2 3 »

45. Salve e feche os arquivos para finalizar a atividade.

Atividade 4 – Menu dinâmico

Objetivos:

- Criar um menu dinâmico com base nas permissões do usuário

Ao navegar na aplicação a experiência do usuários deve ser a mais positivas possível, então com um código simples, vamos adequar o menu e oferecer as opções mais condizentes com cada tipo de usuário.

46. Modifique o arquivo ‘partials/_menu.html’ para que dinamicamente ele reconheça o tipo de usuário, e se for superusuário, tenha a opção de acesso ao admin no menu;

```

<ul>
    {% if user.is_authenticated %}
        <li><a href="{% url 'index' %}">HOME</a></li>
        <li><a href="{% url 'dashboard' %}">Meus pratos</a></li>
        <li><a href="{% url 'cria_prato' %}">Criar pratos</a></li>
        {% if user.is_superuser %}
            <li><a href="{% url 'admin:index' %}">Admin</a></li>
        {% endif %}
        <li><a href="{% url 'logout' %}">Logout</a></li>
    {% else %}
        <li><a href="{% url 'cadastro' %}">Cadastro</a></li>
        <li><a href="{% url 'login' %}">Login</a></li>
    {% endif %}
</ul>

```

47. Para finalizar a atividade e o curso faça todos os testes possíveis, salve e feche os arquivos.

Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extra para rever e aprimorar o que trabalhamos até aqui:

1. *Que tal criar sua própria aplicação ‘do zero’ com um tema bem interessante iniciando um novo projeto empolgante para revisar, fixar e validar o seu conhecimento!*

Deixo uma última mensagem nesse material. Que esse não seja o último desafio, que sigam sempre aprendendo, criando e evoluindo. Criando um futuro melhor!

Desde já agradeço a atenção, respeito e confiança no nosso trabalho.