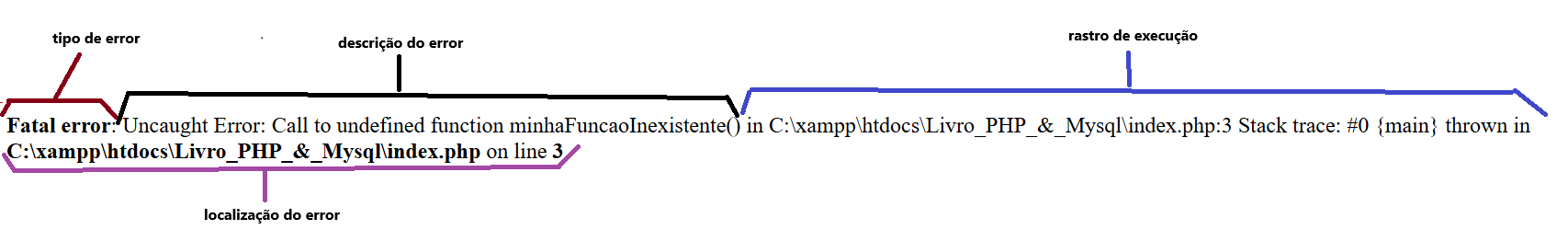
Infelizmente, não importa quão cuidadosos somos escrevendo nossos códigos, erros são fatos da vida. O PHP reportará erros, advertências e avisos para muitos problemas comuns de codificação e problemas em tempo de execução, e saber como detectar e lidar com estes erros farão o processo de depuração muito mais fácil.

# O Básico

Quando ocorre um erro no PHP, ele nos apresenta o erro em uma estrutura para podemos identificar o que é aquele erro e o que o causou, separando a informação da seguinte forma:

## Estrutura de Um Erro no PHP

1. **Tipo de erro:** Primeiro aparece o tipo de erro
   * **Warning**
   * **Fatal error**
   * **Notice**
   * **Parse Error**
   * **etc**
2. **Descrição**: Explica o que deu errado, como;
   * **Undefined Variable**
   * **funtion not found**
   * **etc**
3. **Rastro de execução:** Detalha a sequência de eventos que levou ao erro.
4. **Arquivo e Linha**: Mostra em qual arquivo e em qual linha o erro ocorreu.

**Exemplos da estrutura do erro:**

1. Primeiro vemos o tipo de error, sendo esse um “Fatal error” ou Erro Fatal.
2. Na descrição do error, teremos escrito “Uncaught Error” que signfica que o erro não foi "capturado" por um bloco try-catch. Quando um erro é capturado, ele é tratado de uma maneira que permite que o script continue executando ou tome alguma ação específica.
3. No rastro de execução (stack trace), temos a conjunção “in” entre a descrição e o rastro, que liga gramaticalmente a descrição ao rastro, e logo em seguida o caminho que o PHP percorreu até o erro. O stack trace é um rastro da execução do código que mostra a sequência de chamadas que levou ao erro. Ele ajuda os desenvolvedores a entender como o script chegou ao ponto onde o erro aconteceu. Pense no stack trace como uma "trilha" que o PHP deixa, mostrando por quais funções ou scripts o código passou antes de encontrar o erro.  
   1. Dentro do stack trace temos o: **caminho** +
   2. **#0 {main}**:  
      1. **#0**: Refere-se ao primeiro e único nível na pilha de execução (stack) neste caso. Como o erro ocorreu diretamente no arquivo principal (o script que foi executado), não há outros níveis na pilha. Se o erro tivesse ocorrido dentro de uma função chamada por outra função, você veria mais níveis, como **#1**, **#2**, etc., mostrando as chamadas anteriores.
      2. **{main}**: Refere-se ao "escopo principal" do script, ou seja, o ponto de entrada do código que está sendo executado. Indica que o erro aconteceu no corpo principal do código, não dentro de uma função ou método.
   3. **thrown in**: (lançado em)  
      1. **thrown**: Indica que o erro foi "lançado" ou gerado naquele ponto específico.
      2. **in**: Conjunção gramatical que liga o stack trace ao caminho do arquivo e linha o erro foi lançado
4. Caminho onde está sendo lançado o erro.

## tipos de error

Os tipos de error podem ser classificados de acordo com a importância e impacto no código:

### **1. Erros Críticos** (Interrompem a Execução do Script)

Estes erros são extremamente graves e fazem com que o script pare imediatamente, impossibilitando a continuação da execução.

**Core Error** (Erro de Núcleo)

* Erros fatais que ocorrem durante a inicialização do PHP.

**Parse Error** (Erro de Análise)

* Ocorre quando há um problema de sintaxe no código que impede a análise correta.
* Exemplo: Um ponto e vírgula ausente.

**Fatal Error** (Erro Fatal)

* Um erro crítico que faz o script parar de executar imediatamente.
* Exemplo: Tentar instanciar uma classe inexistente.

**Compile Error** (Erro de Compilação)

* Erros fatais gerados pelo compilador do PHP.

**User Error** (Erro do Usuário)

* Erros fatais gerados pelo próprio usuário usando trigger\_error().

### **2. Erros Recuperáveis** (Podem Ser Tratados, Mas São Graves)

Estes erros são sérios, mas podem ser capturados e tratados para evitar a interrupção do script.

**Recoverable Error** (Erro Recuperável)

* Erros fatais que podem ser capturados e tratados, sem interromper a execução.

### **3. Avisos Importantes** (Não Interrompem o Script, Mas Indicam Problemas Sérios)

Estes avisos indicam problemas que podem afetar a execução do código, mas o script continua rodando.

**Core Warning** (Aviso de Núcleo)

* Avisos que ocorrem durante a inicialização do PHP.

**Warning** (Aviso)

* Um problema que não para o script, mas avisa que algo está errado.
* Exemplo: Incluir um arquivo que não existe com include.

**User Warning** (Aviso do Usuário)

* Avisos gerados pelo usuário usando trigger\_error().

**Compile Warning** (Aviso de Compilação)

* Avisos gerados durante a compilação do código.

### **4. Avisos Leves** (Não Críticos, Mas Requerem Atenção)

Estes avisos apontam para problemas menores que não afetam imediatamente a execução do script, mas que podem ser sinais de lógica incorreta ou uso inadequado.

**Strict** (Aviso Estrito)

* Avisos que recomendam boas práticas para garantir a compatibilidade futura.

**User Notice** (Aviso Simples do Usuário)

* Avisos gerados pelo usuário usando trigger\_error().

**Notice** (Aviso Simples)

* Um erro leve que indica algo potencialmente problemático, mas que não interrompe a execução.
* Exemplo: Usar uma variável que não foi definida.

### **5. Avisos de Depreciação** (Para Futuras Mudanças)

Estes avisos alertam sobre funcionalidades que podem ser removidas em versões futuras e ajudam a manter o código atualizado.

**User Deprecated** (Obsoleto pelo Usuário)

* Avisos gerados pelo usuário sobre funcionalidades obsoletas.

**Deprecated** (Obsoleto)

* Avisos sobre o uso de funcionalidades que podem ser removidas em versões futuras.

# ERROS NO GERAL

Na programação, existem diversos erros que podem ser lançados e outro que são perceptíveis apenas pelo programador. Em geral, todos os erros na programação são classificados como os seguintes tipos:

**1. Erros de Sintaxe**

* **O Que São**: Ocorrem quando o código não segue as regras de sintaxe da linguagem de programação. É como se você tivesse escrito uma frase com a gramática incorreta.
* **Exemplos**:
  + Esquecer um ponto e vírgula (;) no final de uma linha.
  + Usar uma chave de fechamento (}) sem uma chave de abertura ({).
* **Impacto**: O código não será executado até que o erro de sintaxe seja corrigido.

**2. Erros de Semântica**

* **O Que São**: A sintaxe do código está correta, mas o significado do código não é o que você pretendia. É como usar as palavras corretas em uma frase, mas a frase não faz sentido.
* **Exemplos**:
  + Usar a variável errada em um cálculo.
  + Tentar realizar uma operação que não faz sentido no contexto (como dividir um número por uma string).

**4. Erros de Tempo de Execução (Runtime Errors)**

* **O Que São**: Ocorrem enquanto o programa está em execução, e são frequentemente causados por condições inesperadas.
* **Exemplos**:
  + Divisão por zero.
  + Acessar um arquivo que não existe.
* **Impacto**: O programa pode parar de funcionar ou falhar ao processar alguma operação.

**3. Erros de Lógica**

* **O Que São**: O código é executado sem problemas de sintaxe ou semântica, mas não faz o que deveria fazer. São erros de raciocínio ou planejamento na lógica do programa.
* **Exemplos**:
  + Usar um operador + em vez de -.
  + Colocar uma condição if errada que causa um comportamento inesperado.
* **Impacto**: O programa funciona, mas o resultado está incorreto.

**6. Erros de Exceção**

* **O Que São**: Condições excepcionais que ocorrem durante a execução do programa e que podem ser tratadas pelo desenvolvedor.
* **Exemplos**:
  + Tentar acessar um índice de array que não existe.
  + Erros de entrada/saída (I/O), como falha ao abrir um arquivo.
* **Impacto**: Se não forem tratados, podem interromper o programa.

**5. Erros de Compilação**

* **O Que São**: Erros encontrados pelo compilador (em linguagens compiladas) que impedem a tradução do código-fonte em código executável.
* **Exemplos**:
  + Erros de sintaxe ou de tipo que são detectados na fase de compilação.
* **Impacto**: O programa não será compilado até que os erros sejam corrigidos.

**11. Erros de Configuração**

* **O Que São**: Ocorrem quando o ambiente ou configuração do programa está incorreta.
* **Exemplos**:
  + Configurações erradas no arquivo de configuração de um servidor.
  + Variáveis de ambiente não definidas corretamente.

**10. Erros de Concurrency (Concorrência)**

* **O Que São**: Problemas que ocorrem em programas que executam múltiplos processos ou threads simultaneamente.
* **Exemplos**:
  + Condições de corrida (race conditions).
  + Deadlocks, onde dois ou mais processos esperam indefinidamente um pelo outro.

**9. Erros de Memória**

* **O Que São**: Problemas relacionados ao uso incorreto da memória.
* **Exemplos**:
  + Acesso a memória fora dos limites (buffer overflow).
  + Vazamentos de memória, onde a memória alocada não é liberada corretamente.

**8. Erros de Overflow e Underflow**

* **Erros de Overflow**: Ocorrem quando um valor excede o limite máximo que pode ser representado por um tipo de dado.
* **Erros de Underflow**: Ocorrem quando um valor é menor que o limite mínimo que pode ser representado por um tipo de dado.

**7. Erros de Tipagem**

* **O Que São**: Erros relacionados ao uso incorreto de tipos de dados.
* **Exemplos**:
  + Tentar somar uma string com um número (em linguagens fortemente tipadas).
  + Passar o tipo errado de argumento para uma função.
* **Impacto**: Pode levar a falhas ou comportamentos inesperados.

# .ini

Nos arquivos do PHP, temos um arquivo chamado PHP .ini

A extensão **.ini** refere-se a arquivos de configuração no formato **INItialization file** (arquivo de inicialização). Esses arquivos são usados para armazenar configurações e preferências de aplicativos ou do sistema operacional.

**O Que é um Arquivo .ini?**

1. **Formato Simples**: Os arquivos **.ini** são simples arquivos de texto estruturados em seções, com **pares de chave-valor** que definem configurações.
2. **Uso Comum**:
   * São amplamente usados para configurar opções de aplicativos e servidores, como o **php.ini** para configurar o comportamento do PHP.
   * Sistemas operacionais como o Windows também utilizam arquivos .ini para armazenar configurações.

Texto

Descrição gerada automaticamente com confiança médiaComo são arquivos de texto para configurações eles foram padronizados da seguinte forma:

* + **[Seção]**: Define um agrupamento de configurações.
  + **chave=valor**: Define uma configuração específica.

Texto

Descrição gerada automaticamenteQuando estamos dentro de uma IDE, podemos acessar essas configurações da linguagem, escrevendo a chave e o valor correspondente para aquela chave.

Texto

Descrição gerada automaticamenteNo php.ini está assim:

Texto

Descrição gerada automaticamenteInterface gráfica do usuário, Texto

Descrição gerada automaticamente

Perceba uma coisa, no antes da chave receber o valor, ela está sem o “ ; ” no começo.

O **“;”** no início de uma linha indica um **comentário** !

**Linhas com “;”**: São comentários e são ignoradas pelo sistema ou aplicativo que lê o arquivo.

**Linhas sem “;”**: Essas linhas contêm configurações reais, onde uma **chave** recebe um **valor**.

Logo, primeiro precisamos habilitar a configuração, para depois sim, modifica-la.

# Configurando os erros

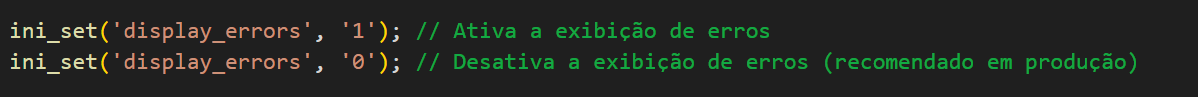
Dentro do PHP.ini temos um montão de diretivas que podem ser configuradas com os valores pré definidos pela linguagem.

Essas diretivas (configurações) podem ser ativadas manualmente diretamente no php.ini ou escritas no código, portanto, apresentaremos 2 exemplos vendo as seguintes diretivas mais usadas.

**display\_errors**

* **Descrição**: Controla se os erros devem ser exibidos na saída padrão (por exemplo, no navegador).
  + On: Exibe erros.
  + Interface gráfica do usuário

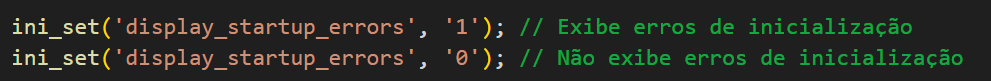
    Descrição gerada automaticamente com confiança médiaOff: Não exibe erros (recomendado em produção).



**display\_startup\_errors**

* **Descrição**: Controla se os erros que ocorrem durante a inicialização do PHP devem ser exibidos.
  + On: Exibe erros de inicialização.
  + Uma imagem contendo Texto

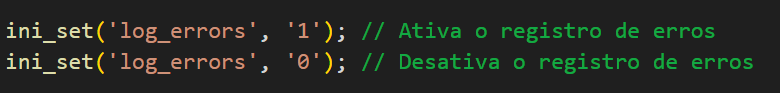
    Descrição gerada automaticamenteOff: Não exibe erros de inicialização (recomendado em produção).



**log\_errors**

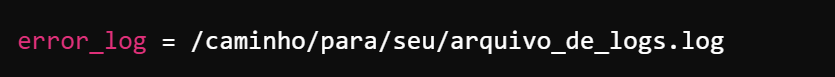
* **Descrição**: Determina se os erros devem ser registrados (logados) em um arquivo ou no sistema de log do servidor.
  + On: Registra os erros.
  + Forma

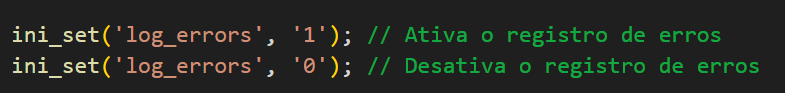
    Descrição gerada automaticamente com confiança médiaOff: Não registra os erros.



**error\_log**

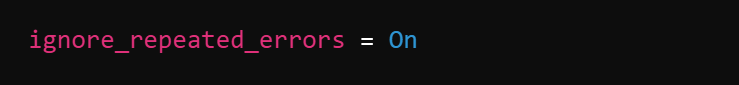
* **Descrição**: Especifica o caminho do arquivo onde os erros devem ser registrados. Se não for definido, os erros são enviados para o log do sistema.

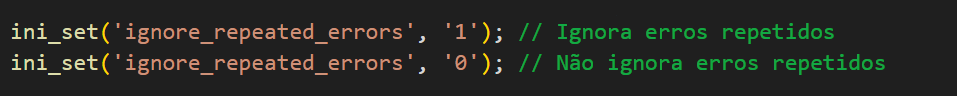




**ignore\_repeated\_errors**

* **Descrição**: Define se erros repetidos devem ser ignorados (não registrados).
  + On: Ignora erros repetidos.
  + Off: Não ignora erros repetidos.



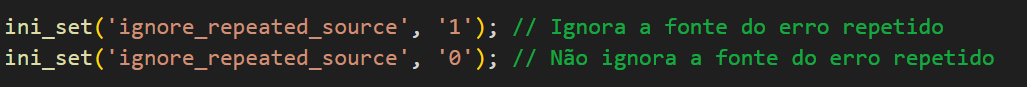


**ignore\_repeated\_source**

* **Descrição**: Define se o PHP deve ignorar erros repetidos do mesmo arquivo e linha de código.
  + On: Ignora a fonte do erro repetido.
  + Off: Não ignora a fonte do erro repetido.

Uma imagem contendo Texto

Descrição gerada automaticamente

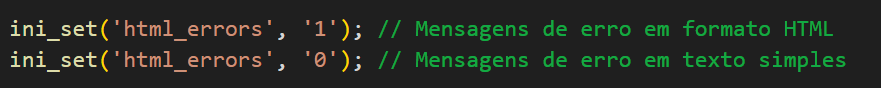


**html\_errors**

* **Descrição**: Controla se as mensagens de erro devem ser formatadas em HTML (útil para depuração em navegadores).
  + On: Mensagens de erro em formato HTML.
  + Off: Mensagens de erro em texto simples.

Interface gráfica do usuário

Descrição gerada automaticamente



**docref\_root** e **docref\_ext**

* **Descrição**: Usadas para gerar links para a documentação do PHP nas mensagens de erro. Isso é útil para depuração.
* **docref\_root**: Especifica o URL da documentação.
* **docref\_ext**: Define a extensão do arquivo da documentação.

Texto

Descrição gerada automaticamente

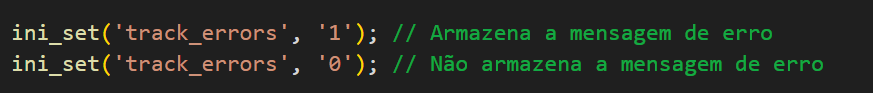




**track\_errors**

* **Descrição**: Se ativado, o PHP armazena a última mensagem de erro em uma variável global chamada $php\_errormsg.
  + On: Armazena a mensagem de erro.
  + Interface gráfica do usuário, Aplicativo

    Descrição gerada automaticamenteOff: Não armazena a mensagem de erro.

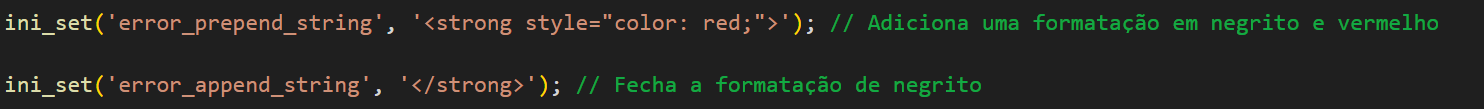


**error\_prepend\_string** & **error\_append\_string**

* **Descrição**: Strings que serão adicionadas antes e depois de uma mensagem de erro.

Uma imagem contendo objeto, relógio, escuro, bola

Descrição gerada automaticamente



# ini\_set

A função ini\_set() no PHP permite que você **configure algumas diretivas do PHP dinamicamente** (ou seja, durante a execução do script). Em outras palavras, ela permite que você altere configurações do PHP temporariamente, sem precisar editar o arquivo **php.ini**

**Limitações do ini\_set()**

* **Nem Todas as Diretivas Podem Ser Alteradas**: Algumas configurações, podem ser restritas por políticas do servidor ou configurações do PHP e só podem ser alteradas manualmente no php.ini.
* **Aplicação Local**: As mudanças feitas com ini\_set() afetam apenas o script atual, e não são persistentes. Quando o script termina, a configuração original do php.ini é restaurada para as próximas execuções.

# Constantes pré definidas & error\_reporting

As **constantes pré-definidas de erro** no PHP são usadas principalmente para definir quais tipos de erros devem ser reportados ou exibidos. Elas funcionam como "valores" que você pode usar para configurar a diretiva **error\_reporting()** ou as configurações de erro no arquivo php.ini.

As constantes pré defindias podem ser usadas tanto dentro de error\_reporting, trigger\_error ou em comparações e condições, veremos a seguir.

**Lista de Constantes Pré-definidas de Erros**

**E\_USER\_ERROR**: Erros fatais gerados pelo usuário com a função trigger\_error().

**E\_USER\_WARNING**: Avisos (não fatais) gerados pelo usuário com trigger\_error().

**E\_USER\_NOTICE**: Avisos (não fatais) gerados pelo usuário com trigger\_error().

**E\_STRICT**: Sugestões para manter o código o mais compatível possível com versões futuras do PHP.

**E\_RECOVERABLE\_ERROR**: Erros fatais que podem ser capturados por um manipulador de exceções (código ainda pode continuar se capturado corretamente).

**E\_DEPRECATED**: Avisos sobre o uso de funcionalidades obsoletas que podem ser removidas em versões futuras.

**E\_USER\_DEPRECATED**: Avisos de funcionalidades obsoletas geradas pelo usuário com trigger\_error().

**E\_ALL**: Constante que representa **todos os erros** e avisos possíveis (exceto **E\_STRICT** em versões anteriores ao PHP 5.4.0).

**E\_ERROR**: Erros fatais de tempo de execução. Estes indicam que o script foi interrompido.

**E\_WARNING**: Avisos de tempo de execução (não fatais). O script não para, mas algo está errado.

**E\_PARSE**: Erros de análise (erros de sintaxe). Estes são gerados apenas pelo parser.

**E\_NOTICE**: Avisos de tempo de execução (não fatais). Indicam algo que pode ser problemático, mas não interrompe o script.

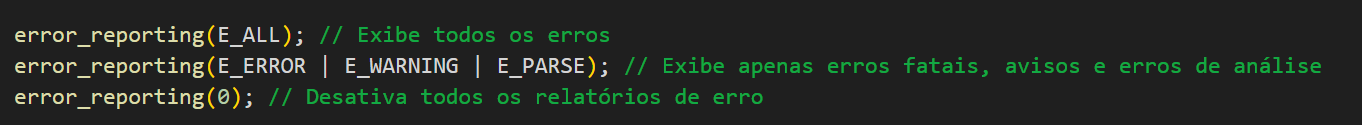
**E\_CORE\_ERROR**: Erros fatais ocorridos durante a inicialização do PHP.

**E\_CORE\_WARNING**: Avisos (não fatais) durante a inicialização do PHP.

**E\_COMPILE\_ERROR**: Erros fatais gerados pelo compilador do PHP.

**E\_COMPILE\_WARNING**: Avisos (não fatais) gerados pelo compilador do PHP.

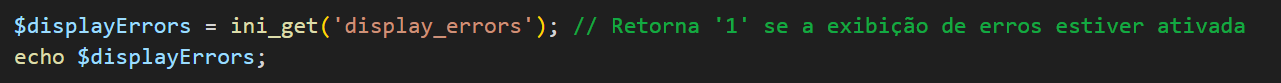
Simplesmente é só escrevermos em nosso código:

error\_reporting( “constante desejada” ).

# Métodos de tratamento de erros

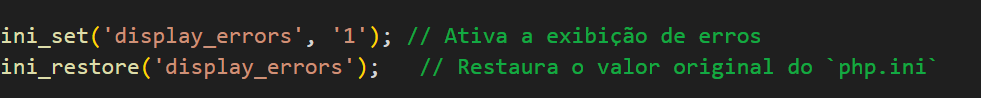
Existem diversos métodos além do ini\_set() para manipulação e acesso as configurações de ambiente e diretivas, iremos abordar apenas algumas aqui, como:

**ini\_get()**

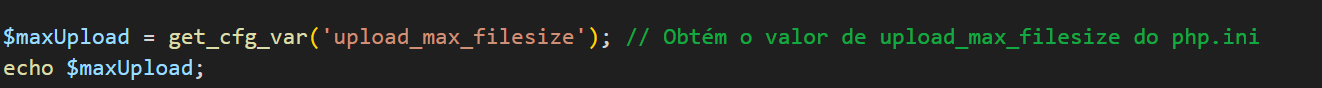
* **Descrição**: Retorna o valor atual de uma diretiva de configuração.
* **Uso**: Use ini\_get() para verificar qual é o valor de uma configuração específica.

**ini\_restore()**

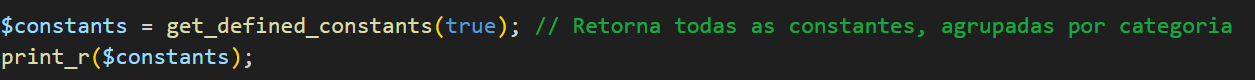
* **Descrição**: Restaura o valor original de uma diretiva, conforme definido no php.ini.
* **Uso**: Use ini\_restore() para reverter uma configuração que foi alterada com



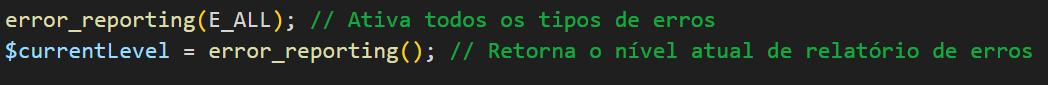
**get\_cfg\_var()**

* **Descrição**: Retorna o valor original de uma diretiva conforme definido no php.ini. Essa função **não leva em consideração as alterações feitas com ini\_set()**.
* **Uso**: Útil para verificar o valor de uma diretiva antes de qualquer alteração em tempo de execução.

**get\_defined\_constants()**

* **Descrição**: Retorna uma lista de todas as constantes definidas, incluindo as constantes de erro, configuração e outras.
* **Uso**: Útil para verificar todas as constantes pré-definidas ou personalizadas.

**error\_reporting()**

* **Descrição**: Define o nível de relatório de erros para o script atual ou retorna o nível atual.
* **Uso**: Você pode usar error\_reporting() para especificar quais tipos de erros devem ser exibidos ou registrados.

**set\_error\_handler()**

* **Descrição**: Define uma função personalizada para manipular erros em vez do manipulador de erros padrão.
* Texto

  Descrição gerada automaticamente**Uso**: Use para capturar e tratar erros de forma personalizada.

**set\_exception\_handler()**

* **Descrição**: Define uma função para manipular exceções não capturadas.
* Texto

  Descrição gerada automaticamente**Uso**: Útil para gerenciar exceções globais de forma personalizada.

**restore\_error\_handler()**

* **Descrição**: Restaura o manipulador de erros padrão do PHP.
* Texto

  Descrição gerada automaticamente**Uso**: Use para voltar ao manipulador de erros padrão depois de usar set\_error\_handler().

**restore\_exception\_handler()**

* **Descrição**: Restaura o manipulador de exceções padrão do PHP.
* **Uso**: Use para voltar ao manipulador de exceções padrão depois de usar set\_exception\_handler().



# personalização da estrutura dos erros

Se você quer capturar e **personalizar erros de forma global**, mesmo aqueles que acontecem fora de um bloco try-catch, você pode usar as seguintes opções:

1. **set\_error\_handler()**: Para tratar **erros de execução**.
2. **set\_exception\_handler()**: Para capturar **exceções não capturadas** globalmente.

**Usando set\_error\_handler() para Personalizar Erros de Execução**

* Tela de celular com aplicativo aberto

  Descrição gerada automaticamente**Descrição**: Essa função define um manipulador de erros personalizado. Você pode usá-la para capturar erros como E\_WARNING, E\_NOTICE, etc.

Saída:

Erro: [8] Undefined variable $variavelNaoDefinida - Arquivo: /caminho/para/seu/arquivo.php, Linha: 7

Perceba que os parâmetros dentro da função nem foi definido, isso por que esses parâmetros já são padrão fornecido do PHP para uso no método set\_error\_handler()

**Parâmetros do Manipulador de Erros**

1. **$errno**: O código ou nível do erro (por exemplo, E\_WARNING, E\_NOTICE, E\_ERROR, etc.). Esse código ajuda a identificar o tipo de erro.
2. **$errstr**: A mensagem de erro, que descreve o que deu errado.
3. **$errfile**: O nome do arquivo no qual o erro ocorreu.
4. **$errline**: O número da linha no arquivo onde o erro foi gerado.

Agora sempre que um error for lançado, ele aparece na estrutura que você definiu.

# Usando HTML e CSS para personalizar o lançamento dos erros

Texto

Descrição gerada automaticamenteTexto

Descrição gerada automaticamenteSimplesmente, podemos fazer isso criando uma estrutura HTML e CSS dentro da própria function, ou se você quiser, carregar dentro da function um arquivo de outro lugar, enfim, agora é a sua imaginção que manda:

**Personalizando Ainda Mais**

Você pode ajustar o HTML e o CSS conforme necessário para se adequar ao seu projeto ou estilo preferido. Por exemplo:

* Adicione ícones de alerta ou imagens.
* Use classes CSS para aplicar estilos de forma mais limpa e reutilizável.
* Crie um layout mais sofisticado, se necessário.

**Dica**

Se você estiver usando um framework ou biblioteca CSS (como Bootstrap), pode usar as classes de estilo prontas para tornar a mensagem de erro ainda mais elegante.

# Registrando em Logs

Em um ambiente de produção, **não é recomendado exibir mensagens de erro detalhadas para os usuários finais** por motivos de segurança e para uma melhor experiência do usuário. Expor mensagens de erro detalhadas pode revelar informações sensíveis sobre a estrutura do seu aplicativo, como caminhos de arquivos, detalhes de configuração do servidor, ou até mesmo vulnerabilidades.

**1. Personalização para Desenvolvedores**

* A personalização da exibição de erros com HTML e CSS, como discutimos, é mais útil em um **ambiente de desenvolvimento**.
* Isso facilita a leitura e depuração dos erros, ajudando os desenvolvedores a identificar rapidamente o problema.
* Em desenvolvimento, ter mensagens de erro detalhadas pode acelerar o processo de correção de bugs.

**2. Prática Padrão em Produção: Registro de Erros em Logs**

* Em um ambiente de produção, a prática padrão é **registrar os erros em um arquivo de log** em vez de exibi-los para o usuário.
* Isso permite que os desenvolvedores revisem os erros posteriormente sem comprometer a segurança da aplicação.

**No Arquivo php.ini**:

Texto

Descrição gerada automaticamente

**No Código PHP**: Se você não tiver acesso ao php.ini, pode configurar isso diretamente no seu script:

Texto

Descrição gerada automaticamente

**Saída:**

[Date and Time] PHP Notice: Undefined variable $variavelNaoDefinida in /caminho/para/seu/script.php on line 6

**Explicando a Saída no Log**:

**Undefined variable $variavelNaoDefinida**: A mensagem de erro que descreve o problema.

**in /caminho/para/seu/script.php on line 6**: O arquivo e a linha onde o erro aconteceu.

**[Date and Time]**: O registro inclui a data e hora em que o erro ocorreu.

**PHP Notice**: O nível do erro (neste caso, um aviso de que a variável não está definida).