Função para Extração e Formatação de Variáveis de Debug

O código abaixo é utilizado para extrair e formatar variáveis de debug de forma controlada em códigos na linguagem python, limitando a profundidade, o tamanho das strings, e evitando referências indesejadas em objetos complexos.

Código:

Python

```
import pprint

def debug_single_object(obj, max_length=20000, string_limit=10, depth=2):
    def is_special_or_function(key, value):
        """Ignora variáveis especiais (__dunder__) e funções."""
        return key.startswith('__') or callable(value)

    def truncate_string(value):
        """Trunca strings longas."""
        return value[:string_limit] + "..." if isinstance(value, str) and
len(value) > string_limit else value

    def extract_object_attrs(obj, level=0):
        """Extrai atributos de objetos sem referências indesejadas."""
        if level >= depth:
            return '...'
        if hasattr(obj, '__dict__'):
            return {
                k: truncate(v, level + 1)
                for k, v in vars(obj).items()
                if not is_special_or_function(k, v)
            }
        return str(obj)  # Retorna representação de string para objetos sem
__dict__

    def truncate(value, level=0):
        """Trunca e processa valores de forma controlada."""
        if level >= depth:  # Limita a profundidade da análise
            return '...'
        if isinstance(value, str):  # Trunca strings
            return truncate_string(value)
        if isinstance(value, (list, tuple)) and len(value) > 10:  # Trunca
listas e tuplas
            truncated = value[:10] + (['...'] if isinstance(value, list)
else ('...',))
```

```
            return truncated if isinstance(value, list) else
tuple(truncated)
        if isinstance(value, dict):  # Analisa dicionários recursivamente
            return {k: truncate(v, level + 1) for k, v in value.items()}
        if not isinstance(value, (int, float, str, list, tuple, dict)):  #
Trata objetos
            return extract_object_attrs(value, level)
        return value  # Retorna outros tipos sem alteração

    # Processa o objeto
    processed_data = truncate(obj)

    # Formata e limita o tamanho da saída
    pretty_output = pprint.pformat(processed_data, depth=depth)
    if len(pretty_output) > max_length:
        return pretty_output[:max_length] + "\n[OUTPUT TRUNCATED]"
    return pretty_output


debug_output = debug_single_object(args.solidity_files)
print(debug_output)
```

Como Utilizar:

1. **Execução:** O código deve estar rodando em modo de debug.
2. **Console de Debug:** No console, passe o objeto que deseja inspecionar como parâmetro da função `debug_single_object`. Exemplo:

```
Python
debug_single_object(objeto_aqui)
```

3. **Saída:** A função retornará uma versão formatada e controlada do objeto, exibida no console ou no terminal.

Funcionalidades do Código:

- **Ignora variáveis especiais e funções:** As variáveis começando com `__` e funções são excluídas para facilitar a leitura.
- **Controle de profundidade:** Evita loops infinitos em estruturas complexas.
- **Truncamento:** Strings, listas e tuplas são limitadas para evitar saídas muito extensas.
- **Formatação legível:** Utiliza `pprint` para exibir a saída de forma organizada.

Essa abordagem é ideal para analisar objetos grandes ou complexos durante o processo de debug.