```
import json
from typing import List, Union, Dict

from pydantic import BaseModel

from swarms.tools.pydantic_to_json import (
    base_model_to_openai_function,
    multi_base_model_to_openai_function,
)


def json_str_to_json(json_str: str) -> dict:
    """Convert a JSON string to a JSON object"""
    return json.loads(json_str)


def json_str_to_pydantic_model(
    json_str: str, model: BaseModel
) -> BaseModel:
    """Convert a JSON string to a Pydantic model"""
    return model.model_validate_json(json_str)


def json_str_to_dict(json_str: str) -> dict:
    """Convert a JSON string to a dictionary"""
    return json.loads(json_str)
```

```
def pydantic_model_to_json_str(
    model: BaseModel, indent: int, *args, **kwargs
) -> str:
    """
    Converts a Pydantic model to a JSON string.

    Args:
        model (BaseModel): The Pydantic model to convert.
        indent (int): The number of spaces to use for indentation.
        *args: Additional positional arguments to pass to `json.dumps`.
        **kwargs: Additional keyword arguments to pass to `json.dumps`.

    Returns:
        str: The JSON string representation of the Pydantic model.
    """
    return json.dumps(
        base_model_to_openai_function(model),
        indent=indent,
        *args,
        **kwargs,
    )


def dict_to_json_str(dictionary: dict) -> str:
```

```
    """Convert a dictionary to a JSON string"""
    return json.dumps(dictionary)


def dict_to_pydantic_model(
    dictionary: dict, model: BaseModel
) -> BaseModel:
    """Convert a dictionary to a Pydantic model"""
    return model.model_validate_json(dictionary)


# def prep_pydantic_model_for_str(model: BaseModel):
#     # Convert to Function
#     out = pydantic_model_to_json_str(model)

#     # return function_to_str(out)


def tool_schema_to_str(
    tool_schema: BaseModel = None, *args, **kwargs
) -> str:
    """Convert a tool schema to a string"""
    out = base_model_to_openai_function(tool_schema)
    return str(out)
```

```
def tool_schemas_to_str(
    tool_schemas: List[BaseModel] = None, *args, **kwargs
) -> str:
    """Convert a list of tool schemas to a string"""
    out = multi_base_model_to_openai_function(tool_schemas)
    return str(out)


def str_to_pydantic_model(string: str, model: BaseModel) -> BaseModel:
    """Convert a string to a Pydantic model"""
    return model.model_validate_json(string)


def list_str_to_pydantic_model(
    list_str: List[str], model: BaseModel
) -> BaseModel:
    """Convert a list of strings to a Pydantic model.

    Args:
        list_str (List[str]): The list of strings to be converted.
        model (BaseModel): The Pydantic model to convert the strings to.

    Returns:
        BaseModel: The Pydantic model with the converted strings.

    """
```

```
    for string in list_str:
        return model.model_validate_json(string)


def prepare_output_for_output_model(
    output_type: Union[str, Dict, BaseModel],
    output: Union[str, Dict, BaseModel] = None,
) -> Union[BaseModel, str]:
    """Prepare the output for the output model.

    Args:
        output_type (Union[str, Dict, BaseModel]): The type of the output.
        output (Union[str, Dict, BaseModel], optional): The output data. Defaults to None.

    Returns:
        Union[BaseModel, str]: The prepared output.

    """
    if output_type == BaseModel:
        return str_to_pydantic_model(output, output_type)
    elif output_type == dict:
        return dict_to_json_str(output)
    elif output_type == str:
        return output
    else:
        return output
```