### Swarms Tool Documentation

A tool is a Python function designed to perform specific tasks, with clear type annotations and comprehensive docstrings. Below are examples of tools to help you get started.

# Rules

To create a tool in the Swarms environment, follow these rules:

1. **Function Definition**:
   - The tool must be defined as a Python function.
   - The function should perform a specific task and be named appropriately.

2. **Type Annotations**:
   - All arguments and the return value must have type annotations.
   - Both input and output types must be strings (`str`).

3. **Docstrings**:
   - Each function must include a comprehensive docstring that adheres to PEP 257 standards. The docstring should explain:
     - The purpose of the function.
     - Arguments: names, types, and descriptions.
     - Return value: type and description.
     - Potential exceptions that the function may raise.

4. **Input and Output Types**:

  - The function's input must be a string.

  - The function's output must be a string.

### Example Tools

### Examples and Anti-Examples

#### Example 1: Fetch Financial News

Correct Implementation

```python
import requests
import os

def fetch_financial_news(query: str = "Nvidia news", num_articles: int = 5) -> str:
    """
    Fetches financial news from the Google News API and returns a formatted string of the top news.

    Args:
        query (str): The query term to search for news. Default is "Nvidia news".
        num_articles (int): The number of top articles to fetch. Default is 5.

    Returns:
```

```
    str: A formatted string of the top financial news articles.


Raises:
    ValueError: If the API response is invalid or there are no articles found.
    requests.exceptions.RequestException: If there is an error with the request.
"""
url = "https://newsapi.org/v2/everything"
params = {
    "q": query,
    "apiKey": os.getenv("NEWSAPI_KEY"),
    "pageSize": num_articles,
    "sortBy": "relevancy",
}


try:
    response = requests.get(url, params=params)
    response.raise_for_status()
    data = response.json()


    if "articles" not in data or len(data["articles"]) == 0:
        raise ValueError("No articles found or invalid API response.")


    articles = data["articles"]
    formatted_articles = []


    for i, article in enumerate(articles, start=1):
            title = article.get("title", "No Title")
            description = article.get("description", "No Description")
            url = article.get("url", "No URL")
            formatted_articles.append(
                f"{i}. {title}\nDescription: {description}\nRead more: {url}\n"
            )

        return "\n".join(formatted_articles)

    except requests.exceptions.RequestException as e:
        print(f"Request Error: {e}")
        raise
    except ValueError as e:
        print(f"Value Error: {e}")
        raise
```

Incorrect Implementation

```python
import requests
import os

def fetch_financial_news(query="Nvidia news", num_articles=5):
    # Fetches financial news from the Google News API and returns a formatted string of the top news.
```

```
url = "https://newsapi.org/v2/everything"
params = {
    "q": query,
    "apiKey": os.getenv("NEWSAPI_KEY"),
    "pageSize": num_articles,
    "sortBy": "relevancy",
}

response = requests.get(url, params=params)
response.raise_for_status()
data = response.json()

if "articles" not in data or len(data["articles"]) == 0:
    raise ValueError("No articles found or invalid API response.")

articles = data["articles"]
formatted_articles = []

for i, article in enumerate(articles, start=1):
    title = article.get("title", "No Title")
    description = article.get("description", "No Description")
    url = article.get("url", "No URL")
    formatted_articles.append(
        f"{i}. {title}\nDescription: {description}\nRead more: {url}\n"
    )
    return "\n".join(formatted_articles)
```

Issues with Incorrect Implementation:

- No type annotations for arguments and return value.

- Missing comprehensive docstring.

#### Example 2: Convert Celsius to Fahrenheit

Correct Implementation

```python
def celsius_to_fahrenheit(celsius_str: str) -> str:
    """
    Converts a temperature from Celsius to Fahrenheit.

    Args:
        celsius_str (str): The temperature in Celsius as a string.

    Returns:
        str: The temperature converted to Fahrenheit as a formatted string.

    Raises:
        ValueError: If the input cannot be converted to a float.
    """
    try:
        celsius = float(celsius_str)
        fahrenheit = celsius * 9/5 + 32
        return f"{celsius}°C is {fahrenheit}°F"
    except ValueError as e:
        print(f"Value Error: {e}")
        raise
```

Incorrect Implementation

```python
def celsius_to_fahrenheit(celsius):
    # Converts a temperature from Celsius to Fahrenheit.
    celsius = float(celsius)
    fahrenheit = celsius * 9/5 + 32
    return f"{celsius}°C is {fahrenheit}°F"
```

Issues with Incorrect Implementation:

- No type annotations for arguments and return value.
- Missing comprehensive docstring.
- Input type is not enforced as string.

#### Example 3: Calculate Compound Interest

Correct Implementation

```python
def calculate_compound_interest(principal_str: str, rate_str: str, time_str: str, n_str: str) -> str:
    """
    Calculates compound interest.

    Args:
        principal_str (str): The initial amount of money as a string.
        rate_str (str): The annual interest rate (decimal) as a string.
        time_str (str): The time the money is invested for in years as a string.
        n_str (str): The number of times that interest is compounded per year as a string.

    Returns:
        str: The amount of money accumulated after n years, including interest.

    Raises:
        ValueError: If any of the inputs cannot be converted to the appropriate type or are negative.
    """
    try:
        principal = float(principal_str)
        rate = float(rate_str)
        time = float(time_str)
        n = int(n_str)

        if principal < 0 or rate < 0 or time < 0 or n < 0:
            raise ValueError("Inputs must be non-negative.")
        amount = principal * (1 + rate / n) ** (n * time)
        return f"The amount after {time} years is {amount:.2f}"
    except ValueError as e:
        print(f"Value Error: {e}")
        raise
```

Incorrect Implementation

```python
def calculate_compound_interest(principal, rate, time, n):
    # Calculates compound interest.
    principal = float(principal)
    rate = float(rate)
    time = float(time)
    n = int(n)

    if principal < 0 or rate < 0 or time < 0 or n < 0:
        raise ValueError("Inputs must be non-negative.")

    amount = principal * (1 + rate / n) ** (n * time)
    return f"The amount after {time} years is {amount:.2f}"
```

Issues with Incorrect Implementation:

- No type annotations for arguments and return value.

- Missing comprehensive docstring.

- Input types are not enforced as strings.

By following these rules and using the examples provided, you can create robust and well-documented tools in the Swarms environment. Ensure that all functions include proper type annotations, comprehensive docstrings, and that both input and output types are strings.

#### Example Tool 4: Reverse a String

Functionality: Reverses a given string.

```python
def reverse_string(s: str) -> str:
    """
    Reverses a given string.

    Args:
        s (str): The string to reverse.

    Returns:
        str: The reversed string.

    Raises:
        TypeError: If the input is not a string.
    """
    try:
        if not isinstance(s, str):
            raise TypeError("Input must be a string.")
        return s[::-1]
    except TypeError as e:
        print(f"Type Error: {e}")
        raise
```

#### Example Tool 5: Check Palindrome

Functionality: Checks if a given string is a palindrome.

```python
def is_palindrome(s: str) -> str:
    """
    Checks if a given string is a palindrome.

    Args:
        s (str): The string to check.

    Returns:
        str: A message indicating whether the string is a palindrome or not.

    Raises:
        TypeError: If the input is not a string.
    """
    try:
        if not isinstance(s, str):
            raise TypeError("Input must be a string.")
        normalized_str = ''.join(filter(str.isalnum, s)).lower()
        is_palindrome = normalized_str == normalized_str[::-1]
        return f"The string '{s}' is {'a palindrome' if is_palindrome else 'not a palindrome'}."
    except TypeError as e:
        print(f"Type Error: {e}")
        raise
```

#### Example Tool 6: Fetch Current Weather

Functionality: Fetches the current weather for a given city from the OpenWeatherMap API.

```python
import requests
import os

def fetch_current_weather(city: str) -> str:
    """
    Fetches the current weather for a given city from the OpenWeatherMap API.

    Args:
        city (str): The name of the city to fetch the weather for.
```

```
Returns:
    str: A formatted string of the current weather in the specified city.

Raises:
    ValueError: If the API response is invalid or the city is not found.
    requests.exceptions.RequestException: If there is an error with the request.
"""
url = "http://api.openweathermap.org/data/2.5/weather"
params = {
    "q": city,
    "appid": os.getenv("OPENWEATHERMAP_KEY"),
    "units": "metric",
}

try:
    response = requests.get(url, params=params)
    response.raise_for_status()
    data = response.json()

    if "weather" not in data or "main" not in data:
        raise ValueError("Invalid API response or city not found.")

    weather_description = data["weather"][0]["description"]
    temperature = data["main"]["temp"]
        return f"The current weather in {city} is {weather_description} with a temperature of
```

```
{temperature}°C."

    except requests.exceptions.RequestException as e:
        print(f"Request Error: {e}")
        raise
    except ValueError as e:
        print(f"Value Error: {e}")
        raise
```

By following the examples provided, you can create your own tools to perform various tasks in the Swarms environment. Ensure each function includes type annotations, comprehensive docstrings, and appropriate error handling to make your tools robust and easy to use.

## Integrate tools into Agent

To integrate tools into an agent, you'd simply just pass in a callable function with types and documentation into the agent class.

```python


from swarms import Agent, OpenAIChat  # ChromaDB
```

```
import subprocess


# Model
llm = OpenAIChat(
    temperature=0.1,
)



# Tools
def terminal(
    code: str,
):
    """
    Run code in the terminal.


    Args:
        code (str): The code to run in the terminal.


    Returns:
        str: The output of the code.
    """
    out = subprocess.run(
        code, shell=True, capture_output=True, text=True
    ).stdout
    return str(out)
```

```
def browser(query: str):
    """
    Search the query in the browser with the `browser` tool.

    Args:
        query (str): The query to search in the browser.

    Returns:
        str: The search results.
    """
    import webbrowser

    url = f"https://www.google.com/search?q={query}"
    webbrowser.open(url)
    return f"Searching for {query} in the browser."


def create_file(file_path: str, content: str):
    """
    Create a file using the file editor tool.

    Args:
        file_path (str): The path to the file.
        content (str): The content to write to the file.
    Returns:
        str: The result of the file creation operation.
    """
    with open(file_path, "w") as file:
        file.write(content)
    return f"File {file_path} created successfully."



def file_editor(file_path: str, mode: str, content: str):
    """
    Edit a file using the file editor tool.

    Args:
        file_path (str): The path to the file.
        mode (str): The mode to open the file in.
        content (str): The content to write to the file.

    Returns:
        str: The result of the file editing operation.
    """
    with open(file_path, mode) as file:
        file.write(content)
    return f"File {file_path} edited successfully."



# Agent
```

```
agent = Agent(
    agent_name="Devin",
    system_prompt=(
        "Autonomous agent that can interact with humans and other"
        " agents. Be Helpful and Kind. Use the tools provided to"
        " assist the user. Return all code in markdown format."
    ),
    llm=llm,
    max_loops="auto",
    autosave=True,
    dashboard=False,
    streaming_on=True,
    verbose=True,
    stopping_token="<DONE>",
    interactive=True,
    tools=[terminal, browser, file_editor, create_file],
    # long_term_memory=chromadb,
    metadata_output_type="json",
    # List of schemas that the agent can handle
    # list_base_models=[tool_schema],
    function_calling_format_type="OpenAI",
    function_calling_type="json",  # or soon yaml
)

# Run the agent
agent.run("Create a new file for a plan to take over the world.")
```

~~~

Example 2

```python
import os

import requests

from swarms import Agent
from swarm_models import OpenAIChat

# Get the OpenAI API key from the environment variable
api_key = os.getenv("OPENAI_API_KEY")

# Create an instance of the OpenAIChat class
model = OpenAIChat(
    api_key=api_key, model_name="gpt-4o-mini", temperature=0.1
)

def fetch_financial_news(
~~~

```
    query: str = "Nvidia news", num_articles: int = 5
) -> str:
    """
    Fetches financial news from the Google News API and returns a formatted string of the top news.

    Args:
        api_key (str): Your Google News API key.
        query (str): The query term to search for news. Default is "financial".
        num_articles (int): The number of top articles to fetch. Default is 5.

    Returns:
        str: A formatted string of the top financial news articles.

    Raises:
        ValueError: If the API response is invalid or there are no articles found.
        requests.exceptions.RequestException: If there is an error with the request.
    """
    url = "https://newsapi.org/v2/everything"
    params = {
        "q": query,
        "apiKey": os.getenv("NEWSAPI_KEY"),
        "pageSize": num_articles,
        "sortBy": "relevancy",
    }

    try:
        response = requests.get(url, params=params)

        response.raise_for_status()

        data = response.json()


        if "articles" not in data or len(data["articles"]) == 0:

            raise ValueError("No articles found or invalid API response.")


        articles = data["articles"]

        formatted_articles = []


        for i, article in enumerate(articles, start=1):

            title = article.get("title", "No Title")

            description = article.get("description", "No Description")

            url = article.get("url", "No URL")

            formatted_articles.append(

                f"{i}. {title}\nDescription: {description}\nRead more: {url}\n"

            )


        return "\n".join(formatted_articles)


    except requests.exceptions.RequestException as e:

        print(f"Request Error: {e}")

        raise

    except ValueError as e:

        print(f"Value Error: {e}")

        raise
```

```
# # Example usage:

# api_key = "ceabc81a7d8f45febfedadb27177f3a3"

# print(fetch_financial_news(api_key))



# Initialize the agent

agent = Agent(

    agent_name="Financial-Analysis-Agent",

    # system_prompt=FINANCIAL_AGENT_SYS_PROMPT,

    llm=model,

    max_loops=2,

    autosave=True,

    # dynamic_temperature_enabled=True,

    dashboard=False,

    verbose=True,

    streaming_on=True,

    # interactive=True, # Set to False to disable interactive mode

    dynamic_temperature_enabled=True,

    saved_state_path="finance_agent.json",

    tools=[fetch_financial_news],

    # stopping_token="Stop!",

    # interactive=True,

    # docs_folder="docs", # Enter your folder name

    # pdf_path="docs/finance_agent.pdf",
    # sop="Calculate the profit for a company.",
    # sop_list=["Calculate the profit for a company."],
    user_name="swarms_corp",
    # # docs=
    # # docs_folder="docs",
    retry_attempts=3,
    # context_length=1000,
    # tool_schema = dict
    context_length=200000,
    # tool_schema=
    # tools
    # agent_ops_on=True,
    # long_term_memory=ChromaDB(docs_folder="artifacts"),
)


# Run the agent
response = agent("What are the latest financial news on Nvidia?")
print(response)


```