



[AutoGen](#)

S

[🏠](#) [Reference](#) [function\\_utils](#)

On this page

# function\_utils

## get\_typed\_annotation

```
def get_typed_annotation(annotation: Any, globalns: Dict[str, Any]) -> Any
```

Get the type annotation of a parameter.

**Arguments:**

- `annotation` - The annotation of the parameter
- `globalns` - The global namespace of the function

**Returns:**

The type annotation of the parameter

## get\_typed\_signature

```
def get_typed_signature(call: Callable[..., Any]) -> inspect.Signature
```

Get the signature of a function with type annotations.

**Arguments:**

- `call` - The function to get the signature for

**Returns:**

The signature of the function with type annotations

## get\_typed\_return\_annotation

```
def get_typed_return_annotation(call: Callable[..., Any]) -> Any
```

Get the return annotation of a function.

**Arguments:**

- `call` - The function to get the return annotation for

**Returns:**

The return annotation of the function

## get\_param\_annotations

```
def get_param_annotations(
    typed_signature: inspect.Signature
) -> Dict[int, Union[Annotated[Type[Any], str], Type[Any]]]
```

Get the type annotations of the parameters of a function

**Arguments:**

- `typed_signature` - The signature of the function with type annotations

**Returns:**

A dictionary of the type annotations of the parameters of the function

## Parameters Objects

```
class Parameters(BaseModel)
```

Parameters of a function as defined by the OpenAI API

## Function Objects

```
class Function(BaseModel)
```

A function as defined by the OpenAI API

## ToolFunction Objects

```
class ToolFunction(BaseModel)
```

A function under tool as defined by the OpenAI API.

### get\_parameter\_json\_schema

```
def get_parameter_json_schema(
    k: str, v: Union[Annotated[Type[Any], str], Type[Any]],
    default_values: Dict[str, Any]) -> JsonSchemaValue
```

Get a JSON schema for a parameter as defined by the OpenAI API

#### Arguments:

- `k` - The name of the parameter
- `v` - The type of the parameter
- `default_values` - The default values of the parameters of the function

#### Returns:

A Pydanitc model for the parameter

### get\_required\_params

```
def get_required_params(typed_signature: inspect.Signature) -> List[str]
```

Get the required parameters of a function

#### Arguments:

- `signature` - The signature of the function as returned by `inspect.signature`

#### Returns:

A list of the required parameters of the function

### get\_default\_values

```
def get_default_values(typed_signature: inspect.Signature) -> Dict[str, Any]
```

Get default values of parameters of a function

#### Arguments:

- `signature` - The signature of the function as returned by `inspect.signature`

#### Returns:

A dictionary of the default values of the parameters of the function

### get\_parameters

```
def get_parameters(required: List[str],
    param_annotations: Dict[str, Union[Annotated[Type[Any],
                                                str],
                                        Type[Any]]],
    default_values: Dict[str, Any]) -> Parameters
```

Get the parameters of a function as defined by the OpenAI API

### Arguments:

- `required` - The required parameters of the function
- `hints` - The type hints of the function as returned by `typing.get_type_hints`

### Returns:

A Pydantic model for the parameters of the function

### `get_missing_annotations`

```
def get_missing_annotations(typed_signature: inspect.Signature,  
                           required: List[str]) -> Tuple[Set[str], Set[str]]
```

Get the missing annotations of a function

Ignores the parameters with default values as they are not required to be annotated, but logs a warning.

### Arguments:

- `typed_signature` - The signature of the function with type annotations
- `required` - The required parameters of the function

### Returns:

A set of the missing annotations of the function

### `get_function_schema`

```
def get_function_schema(f: Callable[..., Any],  
                       *,  
                       name: Optional[str] = None,  
                       description: str) -> Dict[str, Any]
```

Get a JSON schema for a function as defined by the OpenAI API

### Arguments:

- `f` - The function to get the JSON schema for
- `name` - The name of the function
- `description` - The description of the function

### Returns:

A JSON schema for the function

### Raises:

- `TypeError` - If the function is not annotated

### Examples:

```
'''  
def f(a: Annotated[str, "Parameter a"], b: int = 2, c: Annotated[float, "Parameter c"] = 0.1) -> None:  
    pass  
  
get_function_schema(f, description="function f")  
  
# {'type': 'function',  
#  'function': {'description': 'function f',  
#               'name': 'f',  
#               'parameters': {'type': 'object',  
#                              'properties': {'a': {'type': 'str', 'description': 'Parameter a'},  
#                                              'b': {'type': 'int', 'description': 'b'},  
#                                              'c': {'type': 'float', 'description': 'Parameter c'}},  
#                              'required': ['a']}}}  
# '''
```

### `get_load_param_if_needed_function`

```
def get_load_param_if_needed_function(  
    t: Any) -> Optional[Callable[[T, Type[Any]], BaseModel]]
```

Get a function to load a parameter if it is a Pydantic model

### Arguments:

- `t` - The type annotation of the parameter

**Returns:**

A function to load the parameter if it is a Pydantic model, otherwise None

**load\_basemodels\_if\_needed**

```
def load_basemodels_if_needed(func: Callable[..., Any]) -> Callable[..., Any]
```


A decorator to load the parameters of a function if they are Pydantic models

**Arguments:**

- `func` - The function with annotated parameters

**Returns:**

A function that loads the parameters before calling the original function

 [Edit this page](#)

[Previous](#)  
[« code\\_utils](#)

[Next](#)  
[math\\_utils »](#)

Community

[Discord](#) 

[Twitter](#) 