Automate

Writing custom functions

Commercial & Enterprise

Custom functions extend project and deployment capabilities by letting you write Python code to perform specialized extraction, cleaning, validation, and integration. You can use custom functions to calculate values, connect to external APIs, implement complex validation logic, or transform data in ways specific to your business requirements.

Custom functions are available in these variations:

- Custom function fields
- Cleaning functions
- Validation functions
- Deployment integration functions

Supported libraries

The standard Python library is supported, except for the logging service. As an alternative to logging, you can use `print()` statements totaling up to 4 KB in length. If needed, you can split results into multiple print statements.

In addition to the standard Python library, you can import these external libraries at the version indicated.

- datetime 4.4
- dateparser 1.1.4
- fuzzywuzzy 0.18.0
- numpy 1.21.6
- pandas1.5.3
- regex 2022.10.31
- requests 2.25.1
- typing 3.6.2
- typing-extensions 4.5.0

For example:

```
import pandas as pd
print(pd.Series([1,2,3]))
```

Keys enable access to various types of runtime data, configurations, and sensitive values within custom functions.

- **System keys** — Built-in values about files, documents, users, and execution environments available through the `context` parameter. System keys are supported in custom function fields, cleaning functions, and validation functions, but not deployment integration functions.
  - `context['document_text']` — Contains the text content of the current document.
  - `context['file_path']` — Retrieves the path to the uploaded file in the underlying file system. Often used in conjunction with the filesystem API to access file content.
- **Custom keys** — Runtime variables you define for apps and deployments, accessed through the keys parameter using `keys['custom']['<key-name>']`. Values can change between environments. System keys are supported in custom function fields, cleaning functions, validation functions, and deployment integration functions.
- **Secret keys** — Organization-level encrypted values like API keys or credentials, accessed through the keys parameter using `keys['secret']['<key-name>']`. Secrets are managed by administrators and scoped to specific workspaces. System keys are supported in custom function fields, cleaning functions, validation functions, and deployment integration functions.

For custom and secret keys, when writing custom functions in an automation project, you specify *test values* to use during development. After you create an app, you replace test values with runtime values when you run the app.

When you create a deployment based on an app that uses custom or secret keys, you can define default runtime values for the keys. You can't delete keys or modify key names or types that are defined in an app. Keys persist across app versions and remain in the deployment even if modified in newer app versions. In addition to app-specified keys, you can create custom and secret keys specifically for use in deployment integration functions.

When you run a deployment, you can use the runtime values specified in the deployment, or you can override the values.

Custom and secret keys used in app or deployment runs are included in logs. For secret keys, only the key names are displayed, not the actual secret values.

Secrets are automatically redacted when printed to logs with `print(keys['secrets']['my_secret'])`. To bypass this protection, you can use `print(keys['secrets']['my_secret'].get_value())`.

Adding custom and secret keys in custom functions

You can add custom and secret keys in custom functions, including extraction, cleaning, validation, and integration functions.

1. From a custom function editor, click **Show keys** (project editor) or expand the **Keys** section (deployment integration functions).
2. Select the **Custom keys** or **Secret keys** tab, then click to add a key.
3. Specify details about the key:
   - **Key** — Specify a unique name for the key that indicates its purpose.

when you run the app or create a deployment that uses the app.

For secret keys, you can select **None**, choose an organization-level secret available in the workspace, or enter a custom test value.

- **Type** (Custom keys only) — If necessary, modify the custom key value type by clicking the type indicator (**T** by default) and selecting from available options: string, number, boolean, or JSON.
- **Description** — Optionally select the overflow icon •••, then select **Edit description**. Specify details about the key that can help differentiate it and help other users understand its purpose, then click **Save**.

4. (Optional) Close or collapse the **Keys** pane to continue editing the custom function.

Example custom function

Here's an example of a validation function that verifies file size and content.

The example uses a secret key to authorize a call to the filesystem API, retrieving the file using the system key `context['file_path']`. A custom key sets the maximum allowed file size. And the system key `context['document_text']` is used to check that the file isn't empty.

```
def validate_file_size(context, keys):
    """Validate that an uploaded file is under the maximum file size an
    import requests

    API_ROOT = 'https://aihub.instabase.com/api'
    FILES_API_ENDPOINT = API_ROOT + '/v2/files/'

    # Get max file size from max_file_size_mb custom key
    max_size_mb = keys['custom']['max_file_size_mb']

    # Get file using file_path context variable
    url = FILES_API_ENDPOINT + context['file_path']
    params = {'expect-node-type': 'file'}

    # Use api_token secret key for API authentication
    headers = { 'Authorization': f'Bearer {keys["secret"]["api_token"]}

    response = requests.get(url, headers=headers, params=params)
    file_size_mb = int(response.headers['Content-Length']) / (1024 * 10
    if file_size_mb > max_size_mb:
        return (f'File size ({file_size_mb:.1f}MB) exceeds max size ({m
```

Was this page helpful? Yes No