

# chrisvalidation

---

## Index

- [chrisvalidation](#)
  - [Index](#)
  - [1. Introduction & Usage](#)
    - [1-1: Overview](#)
    - [1-2: Quick Start](#)
    - [1-3: File Path & Structure](#)
    - [1-4: First Test API](#)
  - [2. Available Validation Rules](#)
    - [accepted](#)
      - [Implementation](#)
    - [accepted\\_if:anotherfield,value,...](#)
      - [Implementation](#)
    - [array](#)
      - [Implementation](#)
    - [bail](#)
      - [Implementation](#)
    - [boolean|bool](#)
      - [Implementation](#)
    - [in:valuelist](#)
      - [Implementation](#)
    - [in\\_array:anotherfield.\\*](#)
    - [integer|int](#)
      - [Implementation](#)
    - [ip](#)
      - [Implementation](#)
    - [ipv4](#)
      - [Implementation](#)
    - [ipv6](#)
      - [Implementation](#)
    - [json](#)
      - [Implementation](#)
    - [max:value{int}](#)
      - [Implementation](#)
    - [min:value{int}](#)
      - [Implementation](#)
    - [not\\_regex:value{regex}](#)
      - [Implementation](#)
    - [nullable](#)
      - [Implementation](#)
    - [regex:value{regex}](#)
      - [Implementation](#)

- required
  - Implementation
- `size:value{int}`
  - Implementation
- `starts_with:foo,bar,...`
- `string|str`
  - Implementation
- 12. Notes and References
  - Notes
  - References

# 1. Introduction & Usage

## 1-1: Overview

chrisvalidation provides a variety of methods to validate incoming application data. The most common way is to use `validate()` on incoming HTTP requests. However, other validation options are also discussed.

It includes many convenient rules and even supports checking if a value is unique in a database table. Each rule is detailed so you can become familiar with all validation features of chrisvalidation.

## 1-2: Quick Start

To quickly experience the power of chrisvalidation, here's a full example validating a form and returning error messages. This gives you a solid overview of how to validate incoming request data:

## 1-3: File Path & Structure

Assuming the following file structure (using Django as an example):

```
--backend/  
|--- backend/  
|   |--- urls.py  
|   |--- setting.py  
|   |--- ....  
|--- api  
|   |--- api.py  
|   |--- urls.py  
|   |--- ....  
|--- function  
|   |--- validate.py (此函數擺放位置!! 當然你可以擺在其他地方，只要注意路徑是對的就好)
```

Note: Only validation logic is shown here, Django details are omitted.

The function/validate.py file should contain the code from [readme](#).

## 1-4: First Test API

Example login API in api/api.py (username: admin, password: 1234):

```
import json  
from rest_framework import status  
from rest_framework.decorators import api_view  
from rest_framework.response import Response  
  
from function.validation import * # <- Import here!  
  
@api_view(["POST"])  
def signin(request):  
    data=validate(json.loads(request.body),{
```

```

        "username": "required|string",
        "password": "required|string"
    },{
        "required": "ERROR_requestdata_not_found",
        "string": "ERROR_requestdata_type_error"
    })

if data["success"]:
    username=data["data"]["username"]
    password=data["data"]["password"]
    if username=="admin":
        if password=="1234":
            return Response({
                "success": True,
                "data": {
                    "token": "user_token",
                    "userid": "1",
                    "permission": "admin",
                    "name": "chris"
                }
            },status.HTTP_200_OK)
        else:
            return Response({
                "success": False,
                "data": "ERROR_password_error"
            },status.HTTP_401_UNAUTHORIZED)
    else:
        return Response({
            "success": False,
            "data": "ERROR_username_error"
        },status.HTTP_401_UNAUTHORIZED)
else:
    return Response({
        "success": False,
        "data": data["error"]
    },status.HTTP_400_BAD_REQUEST)

```

## 2. Available Validation Rules

Here is the list of all available validation rules:

accepted accepted\_if array bail boolean max in interger ip ipv4 ipv6 JSON min not\_regex nullable regex  
required size string

## accepted

The field under validation must be "yes", "on", 1, "1", true, or "true". This is useful for validating "Terms of Service" acceptance or similar fields.

### Implementation

Check according to the given rule.

code:

```
if value not in ["yes", "on", 1, "1", True, "true"]:  
    return seterror(testkey, rulename)
```

---

## accepted\_if:anotherfield,value,...

The field under validation must be "yes", "on", 1, "1", true, or "true" if another field under validation is equal to a specified value. This is useful for validating "Terms of Service" acceptance or similar fields.

### Implementation

Check according to the given rule.

code:

```
if not isinstance(rulevalue, list) or len(rulevalue) != 2:  
    return seterror(testkey, rulename)  
otherkey = rulevalue[0]  
othervalue = rulevalue[1]  
if otherkey in datadict and datadict[otherkey] == othervalue:  
    if value not in ["yes", "on", 1, "1", True, "true"]:  
        return seterror(testkey, rulename)
```

---

## array

The field being validated must be an array (i.e., must be of `list` type).

### Implementation

Check according to the given rule.

code:

```
if not isinstance(value,list):
    return seterror(testkey,rulename)
```

## bail

Stop running validation rules for the field after the first validation failure.

while the bail rule only stops validating a specific field when a validation failure occurs, you can use the fourth parameter **checkall=True** in the function to stop validating all attributes once a single validation failure happens.

example:

```
validate(data={
    "key": 123
},rule={
    "key": "bail|required|string|min:2"
},error={
    "bail": "ERROR_bail",
    "required": "ERROR_required",
    "string": "ERROR_type_string",
    "min": "ERROR_min_length"
},checkall=True)
```

## Implementation

Check according to the given rule.

code:

```
bailstop=False
for testrule in testrulelist:
    if bailstop:
        break

    returndata=test(fullkey,testrule,value)

    if not returndata["check"]:
        check=False
        errordata[fullkey]={}
        errordata[fullkey]
    [returndata["rulename"]]=returndata["errordata"].replace(":key",f"'{fullkey.split(
\".\")[-1]}'")
        if not firsterror:

firsterror=returndata["errordata"].replace(":key",f"'{fullkey.split(\".\")[-1]}'")
    if not checkall:
```

```
        break
    if "bail" in testrulelist:
        bailstop=True
```

## boolean|bool

The field must be able to convert to a boolean. Accepted values: `true`, `false`, `1`, `0`, `"1"`, `"0"`.

### Implementation

Check according to the given rule.

code:

```
if not isinstance(value,bool) and value not in [0,1,"0","1"]:
    return seterror(testkey,rulename)
```

---

## in:valuelist

The field must be included in the given list (comma-separated).

If the value is an array, every item in the array must exist in the given list.

### Implementation

code:

```
allowed=rulevalue.split(",")
if isinstance(value,list):
    for key in value:
        if str(key) not in allowed:
            return seterror(testkey,rulename)
else:
    if str(value) not in allowed:
        return seterror(testkey,rulename)
```

---

## in\_array:anotherfield.\*

The field must exist in the value(s) of another field.

---

## integer|int

The field must be an integer.

This rule does not verify variable type but instead follows PHP's `FILTER_VALIDATE_INT` logic. For numeric checks, combine with the `numeric` rule.

### Implementation

code:

```
if not isinstance(value,int) and not isinstance(value,float):  
    return seterror(testkey,rulename)
```

---

#### ip

The field must be a valid IP address.

### Implementation

code:

```
try:  
    ipaddress.ip_address(value)  
except:  
    return seterror(testkey,rulename)
```

---

#### ipv4

The field must be a valid IPv4 address.

### Implementation

code:

```
try:  
    if not isinstance(ipaddress.ip_address(value), ipaddress.IPv4Address):  
        return seterror(testkey,rulename)  
except:  
    return seterror(testkey,rulename)
```

---

#### ipv6

The field must be a valid IPv6 address.

### Implementation



code:

```
try:
    if not isinstance(ipaddress.ip_address(value), ipaddress.IPv6Address):
        return seterror(testkey,rulename)
except:
    return seterror(testkey,rulename)
```

---

## json

The field must be JSON (i.e., a dictionary).

### Implementation

code:

```
if not isinstance(value,dict):
    return seterror(testkey,rulename)
```

---

## max: *value*{int}

The field must be less than or equal to the given value. For strings, numbers, arrays, and files, this is evaluated using the `checksize` function.

### Implementation

code:

```
size=checksize(value)
try:
    if size==False or int(rulevalue)<size:
        return seterror(testkey,rulename)
except:
    return seterror(testkey,rulename)
```

---

## min: *value*{int}

The field must be greater than or equal to the given value. Applies to strings, numbers, arrays, and files, using `checksize`.

### Implementation

code:

```

size=checksize(value)
try:
    if size==False or size<int(rulevalue):
        return seterror(testkey,rulename)
except:
    return seterror(testkey,rulename)

```

---

**not\_regex:value{regex}**

The field must **not** match the given regular expression.

When using **regex** or **not\_regex** with patterns containing `|`, use a rule array instead of the pipe character.

### Implementation

code:

```

if type(value)!=str:
    return seterror(testkey,rulename)

pattern=rulevalue

if pattern.startswith("/") and pattern.rfind("/")>0:
    lastslash=pattern.rfind("/")
    regexbody=pattern[1:lastslash]
    flags=pattern[lastslash+1:]
    flagval=0
    if "i" in flags:
        flagval|=re.IGNORECASE
    if "m" in flags:
        flagval|=re.MULTILINE
    if "s" in flags:
        flagval|=re.DOTALL
    pattern=regexbody
else:
    flagval=0

try:
    regex=re.compile(pattern,flagval)
except:
    return seterror(testkey,rulename)

if regex.search(value):
    return seterror(testkey,rulename)

```

---

**nullable**

The field may be `null`.

## Implementation

code:

```
if not ("nullable" in testrulelist) and (value==None):  
    # Normal validation rules here  
    # ...
```

---

`regex:value{regex}`

The field must match the given regular expression.

Use array format if the regex contains `|`.

## Implementation

code:

```
if type(value)!=str:  
    return seterror(testkey,rulename)  
  
pattern=rulevalue  
  
if pattern.startswith("/") and pattern.rfind("/")>0:  
    lastslash=pattern.rfind("/")  
    regexbody=pattern[1:lastslash]  
    flags=pattern[lastslash+1:]  
    flagval=0  
    if "i" in flags:  
        flagval|=re.IGNORECASE  
    if "m" in flags:  
        flagval|=re.MULTILINE  
    if "s" in flags:  
        flagval|=re.DOTALL  
    pattern=regexbody  
else:  
    flagval=0  
  
try:  
    regex=re.compile(pattern,flagval)  
except:  
    return seterror(testkey,rulename)  
  
if not regex.search(value):  
    return seterror(testkey,rulename)
```

## required

The field must exist in the input and cannot be empty. A field is considered empty if:

- The value is `null`
- An empty string
- An empty array or object
- An uploaded file with no path

## Implementation

code:

```
if value is None or value=="" or value==[] or value=={}:  
    return seterror(testkey,rulename)
```

---

## size:value{int}

The field must match the given size.

- For strings: character length
- For numbers: numeric value
- For arrays: number of elements
- For files: file size (in kilobytes)

Examples:

code:

```
"title": "size:12"      # String length = 12  
"seats": "integer|size:10" # Number = 10  
"tags": "array|size:5"  # List has 5 elements  
"image": "file|size:512" # File size = 512KB
```

## Implementation

code:

```
def checksize(value):  
    if isinstance(value,str):  
        return len(value)  
    elif isinstance(value,int) or isinstance(value,float):  
        return value  
    elif isinstance(value,list):  
        return len(value)  
    elif isinstance(value,dict) and "size" in value:
```

```
        return value["size"]
    else:
        return False

size=checksize(value)
try:
    if size==False or size<int(rulevalue):
        return seterror(testkey,rulename)
except:
    return seterror(testkey,rulename)
```

---

starts\_with:foo,bar,...

The field must start with one of the specified values.

---

string|str

The field must be a string. To allow `null`, use the `nullable` rule as well.

### Implementation

code:

```
if not isinstance(value,str):
    return seterror(testkey,rulename)
```

## 12. Notes and References

### Notes

this note is write by chatgpt, maybe will have some mistake.

## References

*20250706 v001000002*