Unevaluated Items

New in draft 2019-09

The `unevaluatedItems` keyword is useful mainly when you want to add or disallow extra items to an array.

`unevaluatedItems` applies to any values not evaluated by an `items`, `prefixItems`, or `contains` keyword. Just as `unevaluatedProperties` affects only **properties** in an object, `unevaluatedItems` affects only **items** in an array.

> Watch out! The word "unevaluated" *does not mean* "not evaluated by `items`, `prefixItems`, or `contains`." "Unevaluated" means "not successfully evaluated", or "does not evaluate to true".

Like with `items`, if you set `unevaluatedItems` to `false`, you can disallow extra items in the array.

schema

```
{
  "prefixItems": [
    { "type": "string" }, { "type": "number" }
  ],
  "unevaluatedItems": false
}
```

Here, all the values are evaluated. The schema passes validation.

data

```
["foo", 42]
```

✓ compliant to schema

But here, the schema fails validation because `"unevaluatedItems": false` specifies that no extra values should exist.

data

```
["foo", 42, null]
```

✗ not compliant to schema

Note that `items` doesn't "see inside" any instances of `allOf`, `anyOf`, or `oneOf` in the same subschema. So in this next example, `items` ignores `allOf` and thus fails to validate.

schema

```
{
  "allOf": [{ "prefixItems": [{ "type": "boolean" }, { "type": "string" }] }],
  "items": { "const": 2 }
}
```

data

```
[true, "a", 2]
```

✗ not compliant to schema

But if you replace `items` with `unevaluatedItems`, then the same array validates.

schema

```
{
  "allOf": [{ "prefixItems": [{ "type": "boolean" }, { "type": "string" }] }],
  "unevaluatedItems": { "const": 2 }
}
```

data

```
[true, "a", 2]
```

✓ compliant to schema

You can also make a "half-closed" schema: something useful when you want to keep the first two arguments, but also add more in certain situations. ("Closed" to two arguments in some places, "open" to more arguments when you need it to be.)

schema

```
{
  "$id": "https://example.com/my-tuple",
  "type": "array",
  "prefixItems": [
    { "type": "boolean" },
    { "type": "string" }
  ],

  "$defs": {
    "closed": {
      "$anchor": "closed",
      "$ref": "#",
      "unevaluatedItems": false
    }
  }
}
```

Here the schema is "closed" to two array items. You can then later use `$ref` and add another item like this:

schema

```
{
  "$id": "https://example.com/my-extended-tuple",
  "$ref": "https://example.com/my-tuple",
  "prefixItems": [
    { "type": "boolean" },
    { "type": "string" },
    { "type": "number" }
  ],

  "$defs": {
    "closed": {
      "$anchor": "closed",
      "$ref": "#",
      "unevaluatedItems": false
    }
  }
}
```

Thus, you would reference `my-tuple#closed` when you need only two items and reference `my-tuple#extended` when you need three items.