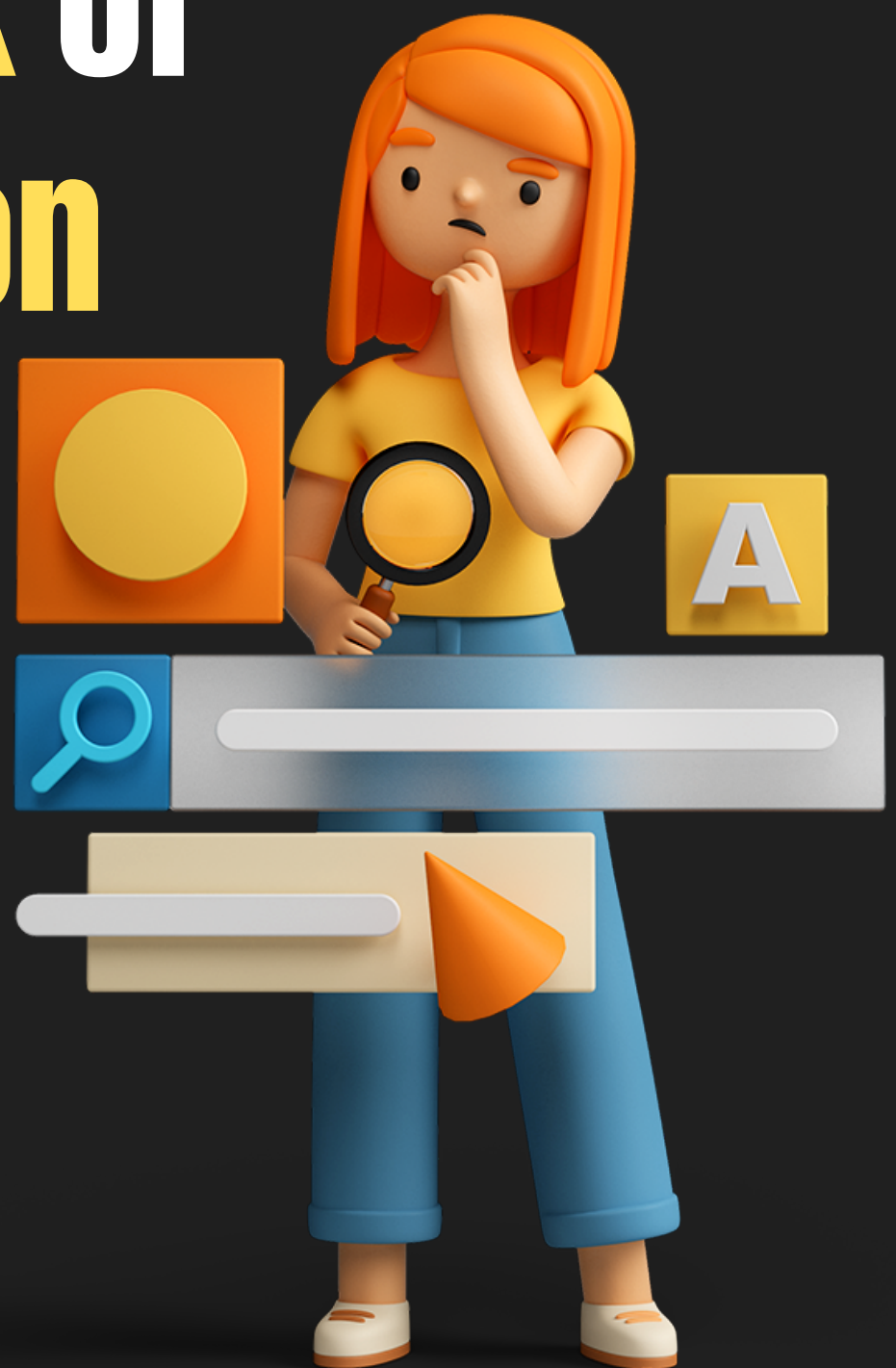
@arjun-panwar

Pydash:

A Kitchen Sink of Missing Python Utilities



Source: https://towardsdatascience.com/pydash-a-bucket-of-missing-python-utilities-5d10365be4fc

FOLLOW US FOR MORE

LIKE, SHARE, SAVE FOR LATER!!

Motivation

Have you ever tried to flatten a nested array like this?

```
a = [ [1,  2,  [4 , 5 ] ] , [6 , 7 ] ]
```

If you found it difficult to flatten such a nested array, you would be happy to find an elegant solution like this:

```
>>> b = [[1, 2, [4, 5]], [6, 7]]

>>> py_.flatten_deep(b)
[1, 2, 4, 5, 6, 7]
```

FOLLOW US FOR MORE
LIKE, SHARE, SAVE FOR LATER!!

@arjun-panwar

...or to get the object of a deeply nested dictionary-like below in one line of code.

```
>>> apple = {
    "price": {
        "in_season": {"store": {"Walmart": [2, 4], "Aldi": 1}},
        "out_of_season": {"store": {"Walmart": [3, 5], "Aldi": 2}},
    }
}

>>> py_.get(apple, "price.in_season.store.Walmart[0]")
2
```

If you are looking for a library that provides useful utilities to deal with Python objects like above, pydash will be your go-to library.

What is pydash?

Pydash is the kitchen sink of Python utility libraries for doing "stuff" in a functional way.
To install pydash, make sure your Python version is >= 3.6, then type:

```
pip install pydash
```

Start with importing pydash:

```
from pydash import py_
```

List

Flatten a Python List

You can flatten a nested Python list using the **py_.flatten** method:

```
>>> a = [[1, 2], [3, 4, 5]]

>>> py_.flatten(a)
[1, 2, 3, 4, 5]
```

What if your list is deeply nested like below?

```
a = [ [1, 2, [4 , 5 ] ] , [6 , 7 ] ]
```

That is when the `py_.flatten` deep method comes in handy:

```
>>> b = [[1, 2, [4, 5]], [6, 7]]

>>> py_.flatten_deep(b)
[1, 2, 4, 5, 6, 7]
```

FOLLOW US FOR MORE
LIKE, SHARE, SAVE FOR LATER!!

@arjun-panwar

Split Elements into Groups

If you can flatten a list, can you also turn a flattened list into a nested one? Yes, that could be done with the **py_.chunk** method:

```
>>> a = [1, 2, 3, 4, 5]

>>> py_.chunk(a, 2)
[[1, 2], [3, 4], [5]]
```

Nice! The elements in the list are split into groups of 2.

Dictionary

Omit Dictionary's Attribute

To omit an attribute from the dictionary, we can use the py_.omit method:

```
>>> fruits = {"name": "apple",
              "color": "red",
              "taste": "sweet"}


>>> py_.omit(fruits, "name")
{'color': 'red', 'taste': 'sweet'}
```

Get Nested Dictionary's Attribute

How do you get the **price** of an **apple** from **Walmart** that is **in season** in a nested dictionary like below?

```
apple = {
"price": {
"in_season": {"store": {"Walmart": [2, 4], "Aldi": 1}},
"out_of_season": {"store": {"Walmart": [3, 5], "Aldi": 2}},
}
}
```

Normally, you need to use a lot of brackets to get that information:

```
apple["price"]["in_season"]["store"]["Walmart"]
```

```
[2,4]
```

FOLLOW US FOR MORE
LIKE, SHARE, SAVE FOR LATER!!

@arjun-panwar

Wouldn't it be nice if you could use the dot notation instead of brackets? That could be done with the py_.get method:

```
>>> py_.get(apple, "price.in_season.store.Walmart")
[2, 4]
```

Cool! You can also get the element in an array using [index] :

```
py_.get(apple, "price.in_season.store.Walmart[0]")
```

```
2
```

List of Dictionaries

Add a subheading

To get the index of an element in a list using a function, use the py_.find_index method:

```
>>> fruits = [
    {"name": "apple", "price": 2},
    {"name": "orange", "price": 2},
    {"name": "grapes", "price": 4},
    ]
>>> filter_fruits = lambda fruit: fruit["name"] == "apple"
>>> py_.find_index(fruits, filter_fruits)
0
```

Find Objects With Matching Style

The py_.find_index method allows you to get the index of the object that matches a certain pattern. But what if you want to get the items in a list instead of the index?

That could be done with the py_.filter method:

```
>>> fruits = [
    {"name": "apple", "price": 2},
    {"name": "orange", "price": 2},
    {"name": "grapes", "price": 4},
    ]

>>> py_.filter_(fruits, {"name": "apple"})
[{'name': 'apple', 'price': 2}]
```

Get Nested Object Value

Sometimes your list of dictionaries can be nested like below. How can you get the taste attribute of apple ?

```
fruits = [
{"name": "apple", "attributes": {"color": "red", "taste": "sweet"}},
{"name": "orange", "attributes": {"color": "orange", "taste": "sweet"}},
{"name": "lemon", "attributes": {"color": "yellow", "taste": "sour"}},
 ]
```

Luckily, this can be easily done with the py_.map_ method:

```
>>> py_.map_(fruits, "attributes.taste")
['sweet', 'sweet', 'sour']
```

FOLLOW US FOR MORE
LIKE, SHARE, SAVE FOR LATER!!

in @arjun-panwar

Execute a Function n Times

You can execute a function n times using the py_.times method. This method is a good alternative to a for loop.

```
>>> py_.times(4, lambda i: f"I have just bought {i} apple")
['I have just bought 0 apple',
 'I have just bought 1 apple',
 'I have just bought 2 apple',
 'I have just bought 3 apple']
```

Chaining

Pydash's Methods

Sometimes you might want to apply several methods to an object. Instead of writing several lines of code, can you apply all methods at once?

That is when method chaining comes in handy. To apply method chaining in an object, use the py_.chain method:

```
>>> fruits = ["apple", "orange", "grapes"]
>>> (
        py_.chain(fruits)
        .without("grapes")
        .reject(lambda fruit: fruit.startswith("a"))
    )
```

FOLLOW US FOR MORE
LIKE, SHARE, SAVE FOR LATER!!

in @arjun-panwar

Note that running the code above will not give us the value:

```
<pydash.chaining.Chain at 0x7f8d256b0dc0>
```

Only when we add .value() to the end of the chain, the final value is computed:

```
>>> fruits = ["apple", "orange", "grapes"]
>>> (
        py_.chain(fruits)
        .without("grapes")
        .reject(lambda fruit: fruit.startswith("a"))
        .value()
    )
['orange']
```

This is called lazy evaluation. Lazy evaluation holds the evaluation of an expression until its value is needed, which avoids repeated evaluation.

Customized Methods

If you want to use your own methods instead of pydash's methods, use the map method:

```
>>> fruits = ["apple", "orange", "grapes"]

>>> def get_price(fruit):
...     prices = {"apple": 2, "orange": 2, "grapes": 4}
...     return prices[fruit]

>>> total_price = py_.chain(fruits).map(get_price).sum()
>>> total_price.value()
8
```

Planting a Value

To replace the initial value of a chain with another value, use the plant method:

```
>>> total_price.plant(["apple", "orange"]).value()
4
```

Cool! We replace `['apple', 'orange', 'grapes']` with `['apple', 'orange']` while using the same chain!

Wanna learn Python, Machine Learning, Deep Learning and Statistics in one-on-one classes

Drop me a message to discuss your requirements

LINKEDIN
linkedin.com/in/arjun-panwar

TELEGRAM
t.me/arjunpanwar

TELEGRAM GROUP
t.me/payperhour