

# Motivation

As the amount of produced libraries, APIs and tools grow dramatically the developers want to reuse them rather than create their own solutions. The fact that such resources as StackOverflow are so popular only proves it. While the most popular questions are covered by these resources, there are still queries that were not answered. It may take from hours to months for a new question to be answered. Also such resources can't help with questions about proprietary libraries. That's why it is necessary to build automatic code search. The labeling of query-code relevance could help to build such a search.

You will be given the pairs of a natural language query and a function or method. You need to label how relevant would be this function or method as the result from search system for the query.

Through the annotations, we want to measure how relevant would these results be to you.

- You don't have to be absolutely certain about the correctness of the code.
- You might be interested in copy-pasting the code, finding a library to use or just getting some understanding about how something is implemented.
- You might be searching within your project (e.g. to reuse code within your project), your company or all of GitHub.

## Instruction for query-function relevance labeling.

If you are **not** provided with the account credentials:

- Register on Labelbox
- Send account's e-mail to the @belatriss (telegram).
- Wait until you are added as a labeler to the project.
- You get an e-mail with the invitation to switch the organisation, do it.
- Go to the projects page – there should appear the project "Query code relevance".
- Start labeling. Label ~ 150-200 pairs. The more pairs you label, the better.

If you are provided with the account credentials:

- Log in Labelbox
- Go to the projects page – there should appear the project "Query code relevance".
- Start labeling. Label ~ 150-200 pairs. The more pairs you label, the better.

## Instruction for labeling.

Please annotate the results according to the following scheme:

- **3: Exact match.** This seems exactly what was looking for. I would copy-paste the code and make minor adaptations or will use this functionality of the library in my code.

Please annotate the following query:

## call external command

```
def call(self, name, *args, **kwargs):
    """Asynchronously call a method of the external environment.

    Args:
        name: Name of the method to call.
        *args: Positional arguments to forward to the method.
        **kwargs: Keyword arguments to forward to the method.

    Returns:
        Promise object that blocks and provides the return value when called.
    """
    payload = name, args, kwargs
    self._conn.send((self._CALL, payload))
    return self._receive
```

[Link to GitHub](#)

0 - Irrelevant   1 - Weak match   2 - Strong match   3 - Exact match   **Skip**

Please annotate the following query:

## Random string generation with upper case letters and digits

```
def gen_password(self, **kw):
    return random_str(5, ''.join([
        string.ascii_uppercase,
        string.digits]), **kw)
```

[Link to GitHub](#)

0 - Irrelevant   1 - Weak match   2 - Strong match   3 - Exact match   **Skip**

- **2: Strong match.** This does more or less what I was looking for. I would use the code in here as a backbone for my purpose, but I won't necessarily copy-paste it or use this library.

Please annotate the following query:

## validate email address

```
def validate_email(self, email):
    """
    Validate the provided email address.

    The email address is first modified to match the RFC spec.
    Namely, the domain portion of the email is lowercased.

    Returns:
        The validated email address.

    Raises:
        serializers.ValidationError:
            If the serializer is bound and the provided email
            doesn't match the existing address.
    """
    user, domain = email.rsplit("@", 1)
    email = "@".join([user, domain.lower()])

    if self.instance and email and self.instance.email != email:
        raise serializers.ValidationError(
            _(
                "Existing emails may not be edited. Create a new one "
                "instead."
            )
        )

    return email
```

[Link to GitHub](#)

0 - Irrelevant

1 - Weak match

2 - Strong match

3 - Exact match

Skip

Please annotate the following query:

## create nested directory

```
def make_dir(path):
    if not os.path.isdir(path):
        logging.info("Creating directory %s" % path)
        os.mkdir(path)
```

[Link to GitHub](#)

0 - Irrelevant

1 - Weak match

2 - Strong match

3 - Exact match

Skip

- **1: Weak match.** That's not exactly what I was looking for, but there are some useful elements/pointers to things that I would use (e.g. APIs, code structure) and can form the basis of a new query or exploration towards solving my query.

Please annotate the following query:

## print coloured text in terminal

```
def print_leaf(leaf, str, index, fd, ctx):  
    if leaf.i_config == True:  
        c = '(rw)'  
        color = optionalconfig  
    else:  
        c = '(ro)'  
        color = notconfig  
  
    m = leaf.search_one('mandatory')  
    if m is None or m.arg == 'false':  
        mand = '?'  
    else:  
        mand = ''  
        color = mandatoryconfig  
    if not index:  
        fd.write("{font: \"Helvetica-Oblique\", color: %s, text: \"%s%s%s %s %s\n\"}" % (color, leaf.arg, str, mand,  
c, get_type_name(leaf)))  
    else:  
        fd.write("{font: \"Helvetica-Oblique\", color: %s, underlined: true, text: \"%s%s%s %s %s\n\"}" % (color, le  
af.arg, str, mand, c, get_type_name(leaf)))
```

[Link to GitHub](#)

0 - Irrelevant   1 - Weak match   2 - Strong match   3 - Exact match   [Skip](#)

- 0: Irrelevant. I would never want to see this for this query.

Please annotate the following query:

## enum value from a string value

```
def value_convert(self, value, value_type):  
    """  
    convert string into type used by `config functions`  
    """  
    CONVERSION_OPTIONS = {  
        "str": str,  
        "int": int,  
        "float": float,  
        # Treat booleans specially  
        "bool": (lambda val: val.lower() in ("true", "1")),  
        # Auto-guess the type  
        "auto": self.make_value,  
    }  
  
    try:  
        return CONVERSION_OPTIONS[value_type](value)  
    except (TypeError, ValueError):  
        self.notify_user("Bad type conversion")  
        return None
```

[Link to GitHub](#)

0 - Irrelevant   1 - Weak match   2 - Strong match   3 - Exact match   [Skip](#)