

# Instruction for query-function relevance labeling.

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.

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](#)

- **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

- **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\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\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](#)