pytoolbox Documentation

Release 14.5.1

David Fischer

Jul 06, 2023

Contents

Contents:

CHAPTER 1

pytoolbox

1.1 pytoolbox package

1.1.1 Subpackages

pytoolbox.ai package

Subpackages

pytoolbox.ai.vision package

Subpackages

pytoolbox.ai.vision.face package

Subpackages

pytoolbox.ai.vision.face.detect package

Submodules

pytoolbox.ai.vision.face.detect.dlib module

pytoolbox.ai.vision.face.recognize package

Submodules

pytoolbox.ai.vision.face.recognize.nn4_small2 module

Submodules

pytoolbox.ai.vision.utils module

pytoolbox.aws package

Submodules

pytoolbox.aws.s3 module

pytoolbox.django package

Subpackages

pytoolbox.django.core package

Submodules

pytoolbox.django.core.constants module

pytoolbox.django.core.exceptions module

pytoolbox.django.core.exceptions.**get_message**(*validation_error*)

pytoolbox.django.core.exceptions.**has_code**(*validation_error*, *code*)

Example usage

```
>>> from django.core.exceptions import ValidationError
>>> has_code(ValidationError('yo'), 'bad')
False
>>> has_code(ValidationError('yo', code='bad'), 'bad')
True
>>> has_code(ValidationError({'__all__': ValidationError('yo')}), 'bad')
False
>>> has_code(ValidationError({'__all__': ValidationError('yo', code='bad')}), 'bad
↪')
True
>>> has_code(ValidationError([ValidationError('yo')]), 'bad')
False
>>> has_code(ValidationError([ValidationError('yo', code='bad')]), 'bad')
True
```

pytoolbox.django.core.exceptions.**iter_validation_errors**(*validation_error*)

Example usage

```
>>> from django.core.exceptions import ValidationError
>>> from pytoolbox.unittest import asserts
>>> eq = lambda i, l: asserts.list_equal(list(i), l)
>>> bad, boy = ValidationError('yo', code='bad'), ValidationError('yo', code='boy
↪')
```

(continues on next page)

(continued from previous page)

```
>>> eq(iter_validation_errors(bad), [(None, bad)])
>>> eq(iter_validation_errors(ValidationError({'__all__': boy})), [('__all__',␣
↪boy)])
>>> eq(iter_validation_errors(ValidationError([bad, boy])), [(None, bad), (None,␣
↪boy)])
```

exception pytoolbox.django.core.exceptions.**DatabaseUpdatePreconditionsError**(*message=None*, ***kwargs*)

Bases: *pytoolbox.exceptions.MessageMixin*, django.db.utils.DatabaseError

message

exception pytoolbox.django.core.exceptions.**InvalidStateError**(*message=None*, ***kwargs*)

Bases: *pytoolbox.exceptions.MessageMixin*, Exception

message

exception pytoolbox.django.core.exceptions.**TransitionNotAllowedError**(*message=None*, ***kwargs*)

Bases: *pytoolbox.exceptions.MessageMixin*, Exception

message

pytoolbox.django.core.validators module

class pytoolbox.django.core.validators.**EmptyValidator**(*regex=None*, *message=None*, *code=None*, *inverse_match=None*, *flags=None*)

Bases: django.core.validators.RegexValidator

regex = '\\S+'

message

code = 'blank'

class pytoolbox.django.core.validators.**KeysValidator**(*required_keys=None*, *optional_keys=None*, *strict=False*, *messages=None*)

Bases: object

A validator designed for HStore to require, even restrict keys.

Code mostly borrowed from:

https://github.com/django/django/blob/master/django/contrib/postgres/validators.py

__init__(*required_keys=None*, *optional_keys=None*, *strict=False*, *messages=None*)

Initialize self. See help(type(self)) for accurate signature.

strict = False

messages

deconstruct()

Return a 3-tuple of class import path, positional arguments, and keyword arguments.

class pytoolbox.django.core.validators.**MD5ChecksumValidator**(*regex=None*, *message=None*, *code=None*, *inverse_match=None*, *flags=None*)

Bases: django.core.validators.RegexValidator

regex = re.compile('[0-9a-f]{32}')

pytoolbox.django.forms package

Submodules

pytoolbox.django.forms.base module

Extra forms.

class pytoolbox.django.forms.base.**SerializedInstanceForm**(***kwargs*)

Bases: object

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

classmethod serialize(*instance*)

instance

is_valid()

pytoolbox.django.forms.fields module

Extra fields for your forms.

class pytoolbox.django.forms.fields.**StripCharField**(***kwargs*)

Bases: django.forms.fields.RegexField

default_widget_attrs = {'autofocus': 'autofocus'}

max_length = None

__init__(***kwargs*)

regex can be either a string or a compiled regular expression object.

pytoolbox.django.forms.mixins module

Mix-ins for building your own forms.

class pytoolbox.django.forms.mixins.**ConvertEmailToTextMixin**(**args*, ***kwargs*)

Bases: object

Set email inputs as text to avoid the i18n issue http://html5doctor.com/html5-forms-input-types#input-email.

__init__(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

class pytoolbox.django.forms.mixins.**EnctypeMixin**

Bases: object

enctype

class pytoolbox.django.forms.mixins.**HelpTextToPlaceholderMixin**(**args*, ***kwargs*)

Bases: object

Update the widgets of the form to copy (and remove) the field's help text to the widget's placeholder.

placeholder_fields = (<class 'django.forms.fields.CharField'>, <class 'django.forms.fi

Add a placeholder to the type of fields listed here.

placeholder_remove_help_text = True

Remove the help text after having copied it to the placeholder.

__init__(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

set_placeholder(*name*, *field*)

class pytoolbox.django.forms.mixins.**MapErrorsMixin**

Bases: object

Map errors based on field name. Mandatory when the form contains a field from a model named differently.

errors_map = {}

add_error(*field*, *error*)

class pytoolbox.django.forms.mixins.**ModelBasedFormCleanupMixin**

Bases: object

Make possible the cleanup of the form by the model through a class method called *clean_form*. Useful to cleanup the form based on complex conditions, e.g. if two fields are inter-related (start/end dates, ...).

clean()

class pytoolbox.django.forms.mixins.**RequestMixin**(**args*, ***kwargs*)

Bases: object

Accept request as a optional (default: None) argument of the constructor and set it as an attribute of the object.

__init__(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

class pytoolbox.django.forms.mixins.**CreatedByMixin**(**args*, ***kwargs*)

Bases: *pytoolbox.django.forms.mixins.RequestMixin*

Set instance's created_by field to current user if the instance is just created.

save(*commit=True*)

class pytoolbox.django.forms.mixins.**StaffOnlyFieldsMixin**(**args*, ***kwargs*)

Bases: *pytoolbox.django.forms.mixins.RequestMixin*

Hide some fields if authenticated user is not a member of the staff.

staff_only_fields = ()

__init__(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

class pytoolbox.django.forms.mixins.**UpdateWidgetAttributeMixin**(**args*, ***kwargs*)

Bases: object

Update the widgets of the form based on a set of rules applied depending of the form field's class. The rules can change the class of the widget and/or update the attributes of the widget with *pytoolbox.django.forms.utils.update_widget_attributes()*.

widgets_rules = {<class 'django.forms.fields.DateField'>: [<class 'pytoolbox.django.f

widgets_common_attrs = {}

Attributes that are applied to all widgets of the form

__init__(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

pytoolbox.django.forms.utils module

Some utilities related to the forms.

pytoolbox.django.forms.utils.**conditional_required**(*form*, *required_dict*, *data=None*, *cleanup=False*)

Toggle requirement of some fields based on a dictionary with 'field name' -> 'required boolean'.

pytoolbox.django.forms.utils.**get_instance**(*form*, *field_name*, *request*, *msg=None*)

Return the instance if the *form* is valid, or try to get it from database. Return None if not found and add an error message if set.

pytoolbox.django.forms.utils.**set_disabled**(*form*, *field_name*, *value=False*)

Toggle the disabled attribute of a form's field.

pytoolbox.django.forms.utils.**update_widget_attributes**(*widget*, *updates*)

Update attributes of a *widget* with content of *updates* handling classes addition [+], removal [-] and toggle [^].

Example usage

```
>>> from pytoolbox.unittest import asserts
>>> widget = type(str(''), (), {})
>>> widget.attrs = {'class': 'mondiale'}
>>> update_widget_attributes(
...     widget, {'class': '+pigeon +pigeon +voyage -mondiale -mondiale, ^voyage ^
↪voyageur'})
>>> asserts.dict_equal(widget.attrs, {'class': 'pigeon voyageur'})
>>> update_widget_attributes(widget, {'class': '+le', 'cols': 100})
>>> asserts.dict_equal(widget.attrs, {'class': 'le pigeon voyageur', 'cols': 100})
```

pytoolbox.django.forms.utils.**validate_start_end**(*form*, *data=None*, *start_name='start_date'*, *end_name='end_date'*)

Check that the field containing the value of the start field (time, ...) is not bigger (>) than the stop.

pytoolbox.django.forms.widgets module

Extra widgets for your forms.

class pytoolbox.django.forms.widgets.**CalendarDateInput**(*attrs=None*, *format=None*)

Bases: django.forms.widgets.DateInput

render(**args*, ***kwargs*)

Render the widget as an HTML string.

media

class pytoolbox.django.forms.widgets.**ClockTimeInput**(*attrs=None*, *format=None*)
Bases: django.forms.widgets.TimeInput

render(**args*, ***kwargs*)
Render the widget as an HTML string.

media

pytoolbox.django.models package

Subpackages

pytoolbox.django.models.fields package

Submodules

pytoolbox.django.models.fields.base module

pytoolbox.django.models.fields.mixins module

Mix-ins for building your own models fields.

class pytoolbox.django.models.fields.mixins.**LowerCaseMixin**
Bases: object

get_prep_value(*value*)

class pytoolbox.django.models.fields.mixins.**OptionsMixin**(***kwargs*)
Bases: object

default_options = {}

override_options = {}

__init__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class pytoolbox.django.models.fields.mixins.**StripMixin**
Bases: object

https://code.djangoproject.com/ticket/6362#no1

default_validators = [<pytoolbox.django.core.validators.EmptyValidator object>]

pre_save(*model_instance*, *add*)

pytoolbox.django.models.managers package

Submodules

pytoolbox.django.models.managers.mixins module

Mix-ins for building your own model managers.

class pytoolbox.django.models.managers.mixins.**AtomicGetUpdateOrCreateMixin**
Bases: object

savepoint = False

get_or_create(*defaults=None*, ***kwargs*)

update_or_create(*defaults=None*, ***kwargs*)

class pytoolbox.django.models.managers.mixins.**AtomicGetRestoreOrCreateMixin**
Bases: object

savepoint = False

get_restore_or_create(**args*, ***kwargs*)

class pytoolbox.django.models.managers.mixins.**CreateModelMethodMixin**
Bases: object

create(**args*, ***kwargs*)

class pytoolbox.django.models.managers.mixins.**StateMixin**
Bases: object

Generate on the fly utility query-set filtering methods to a model using a `pytoolbox.states.StateEnum` to implement its own state machine. Then you can use something like `Model.objects.ready_or_canceled(inverse=True)` to exclude models in state READY or CANCELED.

This mixin requires the following to work:

- Add a *states* attribute to your model class set to the states class you defined earlier.
- Add a *state* field to the model for saving instance state in database.

in_states(*states*, *inverse=False*)
Filter query set to include instances in *states*.

class pytoolbox.django.models.managers.mixins.**RelatedModelMixin**
Bases: object

get_related_manager(*field*)

get_related_model(*field*)

pytoolbox.django.models.query package

Submodules

pytoolbox.django.models.query.mixins module

Mix-ins for building your own query-sets.

class pytoolbox.django.models.query.mixins.**AtomicGetUpdateOrCreateMixin**
Bases: object

savepoint = False

get_or_create(*defaults=None*, ***kwargs*)

update_or_create(*defaults=None*, ***kwargs*)

class pytoolbox.django.models.query.mixins.**AtomicGetRestoreOrCreateMixin**
Bases: object

savepoint = False

get_restore_or_create(**args*, ***kwargs*)

class pytoolbox.django.models.query.mixins.**CreateModelMethodMixin**
Bases: object

create(**args*, ***kwargs*)

class pytoolbox.django.models.query.mixins.**StateMixin**
Bases: object

Generate on the fly utility query-set filtering methods to a model using a `pytoolbox.states.StateEnum` to implement its own state machine. Then you can use something like `Model.objects.ready_or_canceled(inverse=True)` to exclude models in state READY or CANCELED.

This mixin requires the following to work:

- Add a *states* attribute to your model class set to the states class you defined earlier.
- Add a *state* field to the model for saving instance state in database.

in_states(*states*, *inverse=False*)
Filter query set to include instances in *states*.

Submodules

pytoolbox.django.models.base module

pytoolbox.django.models.decorators module

Decorators for enhancing your models.

pytoolbox.django.models.decorators.**with_urls**(*base_url*, **interface_actions*, ***kwargs*)

pytoolbox.django.models.metaclass module

Meta-classes for enhancing your models.

class pytoolbox.django.models.metaclass.**ABCModelMeta**
Bases: abc.ABCMeta, django.db.models.base.ModelBase

Meta-class for building an abstract Model with abstract methods, properties, ...

Example usage

>> class AbstractModel(models.Model): .. __metaclass__ = AbstractModelMeta .. .. class Meta: .. abstract = True

pytoolbox.django.models.mixins module

pytoolbox.django.models.utils module

Some utilities related to the model layer.

pytoolbox.django.models.utils.**get_base_model**(*cls_or_instance*)

pytoolbox.django.models.utils.**get_related_manager**(*cls_or_instance*, *field*)

pytoolbox.django.models.utils.**get_related_model**(*cls_or_instance*, *field*)

pytoolbox.django.models.utils.**get_content_type_dict**(*instance*)
Return a dictionary with the serialized content type and private key of given instance.

pytoolbox.django.models.utils.**get_instance**(*app_label*, *model*, *pk*)
Return an instance given its app_label, model name and private key.

pytoolbox.django.models.utils.**try_get_field**(*instance*, *field_name*)

pytoolbox.django.signals package

Submodules

pytoolbox.django.signals.dispatch module

class pytoolbox.django.signals.dispatch.**InstanceSignal**(*providing_args=None*, *use_caching=False*)
Bases: django.dispatch.dispatcher.Signal

send(*sender=None*, ***named*)
Send signal from sender to all connected receivers.

If any receiver raises an error, the error propagates back through send, terminating the dispatch loop. So it's possible that all receivers won't be called if an error is raised.

Arguments:

sender The sender of the signal. Either a specific object or None.

named Named arguments which will be passed to receivers.

Return a list of tuple pairs [(receiver, response), ...].

send_robust(*sender=None*, ***named*)
Send signal from sender to all connected receivers catching errors.

Arguments:

sender The sender of the signal. Can be any Python object (normally one registered with a connect if you actually want something to occur).

named Named arguments which will be passed to receivers.

Return a list of tuple pairs [(receiver, response), ...].

If any receiver raises an error (specifically any subclass of Exception), return the error instance as the result for that receiver.

pytoolbox.django.signals.handlers module

apps.py

```
from django import apps
from django.utils.translation import gettext_lazy as _

from . import signals

__all__ = ('MyApp', )
```

(continues on next page)

(continued from previous page)

```
class MyAppConfig(apps.AppConfig):
    name = 'myapp'
    verbose_name = _('My Application')

    def ready(self):
        signals.connect(self)
```

signals.py

```
from django.db.models import signals as dj_signals
from django.db.backends import signals as dj_db_signals
from pytoolbox.django.signals import (
    create_site, setup_postgresql_hstore_extension, strip_strings_and_validate_model
)

# ...

def connect(config):
    '''Connect signal handlers to signals.'''
    dj_db_signals.connection_created.connect(setup_postgresql_hstore_extension)
    dj_signals.post_migrate.connect(create_site, sender=config)
    dj_signals.pre_save.connect(
        strip_strings_and_validate_model, sender=settings.AUTH_USER_MODEL)
```

pytoolbox.django.signals.handlers.**clean_files_delete_handler**(*instance*, *signal*, ***kwargs*)

Remove the files of the instance's file fields when it is removed from the database.

Simply use `post_delete.connect(clean_files_delete_handler, sender=<your_model_class>)`

> **Warning:** This function remove the file without worrying about any other instance using this file !

> **Note:** Project django-cleanup is a more complete alternative.

pytoolbox.django.signals.handlers.**create_site**(*sender*, ***kwargs*)

Ensure the site name and domain is well configured.

Some alternative:

- Loading an initial fixture with the values for the site
- The application django-defaultsite
- **Other options discussed here:** here

pytoolbox.django.signals.handlers.**setup_postgresql_hstore_extension**(*sender*, *connection*, ***kwargs*)

pytoolbox.django.signals.handlers.**strip_strings_and_validate_model**(*sender*, *instance*, *raw*, ***kwargs*)

Strip the string fields of the instance and run the instance's full_clean().

pytoolbox.django.test package

Subpackages

pytoolbox.django.test.runner package

Submodules

pytoolbox.django.test.runner.mixins module

Mix-ins for building your own test runners.

class pytoolbox.django.test.runner.mixins.**CeleryInMemoryMixin**
Bases: object

setup_test_environment()

class pytoolbox.django.test.runner.mixins.**FastPasswordHasherMixin**
Bases: object

setup_test_environment()

class pytoolbox.django.test.runner.mixins.**TemporarySendfileRootMixin**
Bases: object

setup_test_environment()

Submodules

pytoolbox.django.test.mixins module

pytoolbox.django.utils package

Submodules

pytoolbox.django.utils.collections module

class pytoolbox.django.utils.collections.**FieldsToValuesLookupDict**(*name*, *translations=None*)
Bases: object

Global registry for mapping X class fields to W values.

- X can be a Model, a (Model)Form, a REST Framework Serializer, ...
- Y can be the fields help texts or verbose names, or the number 42.

Strange idea? Isn't it?

Here is a short example as an appetizer. Suppose you want to define your application's help texts into a centralized registry, for keeping your wording DRY. And suppose you have some models like this:

>> class Media(models.Model): .. url = models.URLField()

>> class File(models.Model): .. url = models.URLField()

And you instantiate this class with:

>> help_texts = FieldsLookupDict({'Media.url': 'The media asset ingest URL', 'url': 'An URL'})

Then, you can lookup for the help text of a field like this:

>> help_texts[(Media, 'url')] The media asset ingest URL

>> help_texts[(File, 'url')] An URL

The value returned will be the first matching to the following keys:

1. '<cls.__name__>.<field_name>'
2. '<field_name>'

If given class have a _meta or Meta ("meta") attribute with a model attribute, then the following keys are tried:

1. '<cls.__name__>.<field_name>'
2. '<cls._meta.model>.<field_name>'
3. '<field_name>'

__init__(*name*, *translations=None*)
Initialize self. See help(type(self)) for accurate signature.

pytoolbox.django.utils.logging module

pytoolbox.django.utils.logging.**log_to_console**(*settings*)
Update settings to make all loggers use the console.

Example usage

```
>>> import collections
>>> settings = collections.namedtuple('settings', ['DEBUG', 'LOGGING'])
>>> settings.DEBUG = True
>>> settings.LOGGING = {
...     'version': 1,
...     'loggers': {
...         'django': {
...             'handlers': ['file'], 'level': 'INFO', 'propagate': True
...         },
...         'django.request': {
...             'handlers': ['mail_admins'], 'level': 'ERROR', 'propagate': True
...         },
...         'mysite': {
...             'handlers': ['console'], 'level': 'INFO', 'propagate': True
...         }
...     }
... }
>>> expected_settings = collections.namedtuple('settings', ['DEBUG', 'LOGGING'])
>>> expected_settings.DEBUG = True
>>> expected_settings.LOGGING = {
...     'version': 1,
...     'loggers': {
...         'django': {
...             'handlers': ['console'], 'level': 'INFO', 'propagate': True
...         },
```

(continues on next page)

(continued from previous page)

```
...         'django.request': {
...             'handlers': ['console'], 'level': 'ERROR', 'propagate': True
...         },
...         'mysite': {
...             'handlers': ['console'], 'level': 'INFO', 'propagate': True
...         }
...     }
... }
>>> log_to_console(settings)
>>> settings.LOGGING == expected_settings.LOGGING
True
```

pytoolbox.django.views package

Submodules

pytoolbox.django.views.base module

Extra views.

class pytoolbox.django.views.base.**CancellableDeleteView**(***kwargs*)
Bases: django.views.generic.edit.DeleteView

Handle the cancel action (detect a cancel parameter in the POST request).

post(*request*, **args*, ***kwargs*)

pytoolbox.django.views.mixins module

Mix-ins for building your own views.

class pytoolbox.django.views.mixins.**AddRequestToFormKwargsMixin**
Bases: object

Add the view request to the keywords arguments for instantiating the form.

get_form_kwargs(**args*, ***kwargs*)

should_add_request_to_form_kwargs()

class pytoolbox.django.views.mixins.**BaseModelMultipleMixin**
Bases: object

get_context_object_name(*instance_list*)
Get the name of the item to be used in the context.

class pytoolbox.django.views.mixins.**BaseModelSingleMixin**
Bases: object

get_context_object_name(*instance*)
Get the name to use for the instance.

class pytoolbox.django.views.mixins.**InitialMixin**
Bases: object

Add helpers to safely use the URL query string to fill a form with initial values.

initials = {}

get_initial()

set_inital(*initial*, *name*, *default*)

set_initial_from_func(*initial*, *name*, *default*, *func*, *msg_value*, *mgs_missing*)

set_initial_from_model(*initial*, *name*, *default*, *model*, *msg_value*, *mgs_missing*)

class pytoolbox.django.views.mixins.**LoggedCookieMixin**
Bases: object

Add a "logged" cookie set to "True" if user is authenticated else to "False".

post(**args*, ***kwargs*)

class pytoolbox.django.views.mixins.**RedirectMixin**
Bases: object

Redirect to a page.

redirect_view = None

dispatch(*request*, **args*, ***kwargs*)

class pytoolbox.django.views.mixins.**TemplateResponseMixin**
Bases: django.views.generic.base.TemplateResponseMixin

default_template_directory = 'default'

get_template_names()
Return a list of template names to be used for the request. Must return a list. May not be called if render_to_response() is overridden.

class pytoolbox.django.views.mixins.**ValidationErrorsMixin**
Bases: object

form_valid(*form*)

pytoolbox.django.views.utils module

Some utilities related to the view layer.

pytoolbox.django.views.utils.**get_model_or_404**(*name*, **models*)

Submodules

pytoolbox.django.storage module

Extra storages and mix-ins for building your own storages.

class pytoolbox.django.storage.**ExpressTemporaryFileMixin**
Bases: object

class pytoolbox.django.storage.**OverwriteMixin**
Bases: object

Update get_available_name to remove any previously stored file (if any) before returning the name.

get_available_name(*name*)

class pytoolbox.django.storage.**OverwriteFileSystemStorage**(*location=None*, *base_url=None*, *file_permissions_mode=None*, *directory_permissions_mode=None*)

Bases: *pytoolbox.django.storage.OverwriteMixin*, django.core.files.storage.FileSystemStorage

A file-system based storage that let overwrite files with the same name.

pytoolbox.django.templatetags module

pytoolbox.django.urls module

pytoolbox.django_datatable_view package

Subpackages

pytoolbox.django_datatable_view.views package

Submodules

pytoolbox.django_datatable_view.views.mixins module

Mix-ins for building your own Django Datatable View powered views.

class pytoolbox.django_datatable_view.views.mixins.**MultiTablesMixin**

Bases: object

Implements the base code for using multiple django-datatable-views powered tables.

multi_default = 'default'

multi_datatables = ()

request_name_key = 'datatable-name'

get_ajax_url(*name=None*)

get_context_data(***kwargs*)

get_datatable(*name=None*)

get_datatable_name(*name*)

get_datatable_structure(*name=None*)

get_queryset(*name=None*)

pytoolbox.django_filter package

Subpackages

pytoolbox.django_filter.filterset package

Submodules

pytoolbox.django_filter.filterset.mixins module

Mix-ins for building your own Django Filter powered filters.

class pytoolbox.django_filter.filterset.mixins.**RaiseOnUnhandledFieldClassMixin**
Bases: object

Raise an exception when the filter set is unable to find a suitable filter for any of the model fields to filter.

classmethod filter_for_field(*f*, *name*, *lookup_type='exact'*)

pytoolbox.django_formtools package

Subpackages

pytoolbox.django_formtools.views package

Submodules

pytoolbox.django_formtools.views.mixins module

Mix-ins for building your own Django Form Tools powered views.

class pytoolbox.django_formtools.views.mixins.**CrispyFormsMixin**
Bases: object

get_context_data(*form*, ***kwargs*)
Add the management form to the form for working with crispy forms.

class pytoolbox.django_formtools.views.mixins.**DataTableViewCompositionMixin**
Bases: object

Compose the wizard with some tables views.

table_view_classes = {}

get(*request*, **args*, ***kwargs*)
Retrieve the table view and delegate AJAX to the table view.

get_context_data(***kwargs*)
Update the context with the context returned by the table view.

get_table_view()
Return an instance of the datatable-view for current step, defaults to None.

class pytoolbox.django_formtools.views.mixins.**SerializeStepInstanceMixin**
Bases: object

serialized_instance_form_class
alias of *pytoolbox.django.forms.base.SerializedInstanceForm*

serialized_instances_key = 'serialized-instances'

serialized_instances

serialize_step_instance(*form*, *step=None*)

get_form(*step=None*, **args*, ***kwargs*)

get_form_kwargs(*step*)

pytoolbox.multimedia package

Subpackages

pytoolbox.multimedia.exif package

Submodules

pytoolbox.multimedia.exif.brand module

pytoolbox.multimedia.exif.camera module

pytoolbox.multimedia.exif.equipment module

pytoolbox.multimedia.exif.image module

pytoolbox.multimedia.exif.lens module

pytoolbox.multimedia.exif.metadata module

pytoolbox.multimedia.exif.photo module

pytoolbox.multimedia.exif.tag module

pytoolbox.multimedia.ffmpeg package

Submodules

pytoolbox.multimedia.ffmpeg.encode module

pytoolbox.multimedia.ffmpeg.ffmpeg module

pytoolbox.multimedia.ffmpeg.ffprobe module

pytoolbox.multimedia.ffmpeg.miscellaneous module

pytoolbox.multimedia.ffmpeg.utils module

pytoolbox.multimedia.image package

Submodules

pytoolbox.multimedia.image.PIL module

pytoolbox.multimedia.image.PIL.**get_orientation**(*image*, *orientation_tag=274*, *no_exif_default=None*, *no_key_default=None*)

pytoolbox.multimedia.image.PIL.**apply_orientation**(*image*, *get_orientation=<function get_orientation>*, *sequences={None: [], 1: [], 2: [0], 3: [3], 4: [1], 5: [0, 2], 6: [4], 7: [1, 2], 8: [2]}*)

Credits: https://stackoverflow.com/questions/4228530/pil-thumbnail-is-rotating-my-image.

pytoolbox.multimedia.image.PIL.**open**(*file_or_path*)

pytoolbox.multimedia.image.PIL.**remove_metadata**(*image*, *keys=('exif',)*, *inplace=False*)

pytoolbox.multimedia.image.PIL.**remove_transparency**(*image*, *background=(255, 255, 255)*)

Return a RGB image with an alpha mask applied to picture + background. If image is already in RGB, then its a no-op.

pytoolbox.multimedia.image.PIL.**save**(*image*, **args*, ***kwargs*)

Submodules

pytoolbox.multimedia.x264 module

pytoolbox.network package

Subpackages

pytoolbox.network.smpte2022 package

Submodules

pytoolbox.network.smpte2022.base module

pytoolbox.network.smpte2022.generator module

pytoolbox.network.smpte2022.receiver module

Submodules

pytoolbox.network.http module

pytoolbox.network.ip module

pytoolbox.network.ip.**IPSocket**(*socket*)

This helper create a dictionary containing address and port from a parsed IP address string. Throws InvalidIPSocketError in case of failure.

Example usage

```
>>> IPSocket('gaga:gogo')
Traceback (most recent call last):
    ...
pytoolbox.exceptions.InvalidIPSocketError: gaga:gogo is not a valid IP socket.
>>>
>>> from pytoolbox.unittest import asserts
>>> asserts.dict_equal(
...     IPSocket('239.232.0.222:5004'),
...     {'ip': '239.232.0.222', 'port': 5004})
```

Warning: TODO IPv6 ready : >>> IPSocket('[2001:0db8:0000:0000:0000:ff00:0042]:8329')

pytoolbox.network.ip.**ip_address**(*address*)

Take an IP string/int and return an object of the correct type.

Args:

address: A string or integer, the IP address. Either IPv4 or IPv6 addresses may be supplied; integers less than 2**32 will be considered to be IPv4 by default.

Returns: An IPv4Address or IPv6Address object.

Raises:

ValueError: if the *address* passed isn't either a v4 or a v6 address

pytoolbox.network.rtp module

class pytoolbox.network.rtp.**RtpPacket**(*data*, *length*)

Bases: object

This represent a real-time transport protocol (RTP) packet.

- **RFC 3550**
- Wikipedia (RTP)
- Parameters (RTP)

Packet header

- RFC 3550 page 13

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                               ....                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Extension header

- RFC 3550 page 19

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      defined by profile       |           length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        header extension                       |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

ER_VERSION = 'RTP Header : Version must be set to 2'

ER_PADDING_LENGTH = 'RTP Header : Bad padding length'

ER_EXTENSION_LENGTH = 'RTP Header : Bad extension length'

ER_PAYLOAD = 'RTP packet must have a payload'

HEADER_LENGTH = 12

V_MASK = 192

V_SHIFT = 6

P_MASK = 32

X_MASK = 16

CC_MASK = 15

M_MASK = 128

PT_MASK = 127

DYNAMIC_PT = 96

MP2T_PT = 33

MP2T_CLK = 90000

S_MASK = 65535

TS_MASK = 4294967295

valid
Returns True if this packet is a valid RTP packet.

validMP2T
Returns True if this packet is a valid RTP packet containing a MPEG2-TS payload.

errors
Returns an array containing any errors.

Returns array of error message(s).

Example usage

Testing invalid header:

```
>>> from pytoolbox.unittest import asserts
>>> rtp = RtpPacket(bytearray(RtpPacket.HEADER_LENGTH-1), RtpPacket.HEADER_
↪LENGTH-1)
>>> asserts.list_equal(rtp.errors, [
...     'RTP Header : Version must be set to 2',
```

(continues on next page)

(continued from previous page)

```
...     'RTP packet must have a payload'
... ])
```

Testing a valid RTP packet with a MPEG2-TS payload:

```
>>> rtp = RtpPacket.create(6, 777, RtpPacket.MP2T_PT, 'salut')
>>> asserts.list_equal(rtp.errors, [])
```

clock_rate
Return the MPEG2-TS clock rate of a MPEG2-TS payload or 1 if this is not.

header_size
Returns the length (aka size) of the header.

Example usage

```
>>> rtp = RtpPacket.create(6, 777, RtpPacket.MP2T_PT, 'salut')
>>> print(rtp.header_size)
12
```

payload_size
Returns the length (aka size) of the payload.

Example usage

```
>>> rtp = RtpPacket.create(6, 777, RtpPacket.MP2T_PT, 'salut')
>>> print(rtp.payload_size)
5
```

time
Return computed time (*timestamp / clock rate*).

header_bytes
Return the RTP header bytes.

Example usage

```
>>> rtp = RtpPacket.create(6, 777, RtpPacket.MP2T_PT, bytearray.fromhex('00␣
↪01 02 03'))
>>> print(rtp)
version      = 2
errors       = []
padding      = False
extension    = False
marker       = False
payload type = 33
sequence     = 6
timestamp    = 777
clock rate   = 90000
time         = 0
ssrc         = 0
csrc count   = 0
payload size = 4
>>> header = rtp.header_bytes
>>> assert len(header) == 12
>>> print(''.join(' %02x' % b for b in header))
 80 21 00 06 00 00 03 09 00 00 00 00
```

(continues on next page)

(continued from previous page)

```
>>> header += rtp.payload
>>> assert rtp == RtpPacket(header, len(header))
```

```
>>> rtp = RtpPacket.create(0xffffffff, 0xffffffffff, RtpPacket.DYNAMIC_PT,␣
↪bytearray(1023))
>>> print(rtp)
version      = 2
errors       = []
padding      = False
extension    = False
marker       = False
payload type = 96
sequence     = 65535
timestamp    = 4294967295
clock rate   = 1
time         = 4294967295
ssrc         = 0
csrc count   = 0
payload size = 1023
>>> header = rtp.header_bytes
>>> assert len(header) == 12
>>> print(''.join(' %02x' % b for b in header))
 80 60 ff ff ff ff ff ff 00 00 00 00
>>> header += rtp.payload
>>> assert rtp == RtpPacket(header, len(header))
```

bytes

Return the RTP packet header and payload bytes.

__init__(*data*, *length*)

This constructor will parse input bytes array to fill packet's fields. In case of error (e.g. bad version number) the constructor will abort filling fields and un-updated fields are set to their corresponding default value.

Parameters

- **bytes** (*bytearray*) – Input array of bytes to parse as a RTP packet
- **length** (*int*) – Amount of bytes to read from the array of bytes

Example usage

Testing invalid headers:

```
>>> rtp = RtpPacket(bytearray(RtpPacket.HEADER_LENGTH-1), RtpPacket.HEADER_
↪LENGTH-1)
>>> rtp.valid  # Bad length
False
>>> rtp = RtpPacket(bytearray(RtpPacket.HEADER_LENGTH), RtpPacket.HEADER_
↪LENGTH)
>>> rtp.valid  # Bad version
False
>>> bytes = bytearray(RtpPacket.HEADER_LENGTH)
>>> bytes[0] = 0xa0
>>> rtp = RtpPacket(bytes, RtpPacket.HEADER_LENGTH)
>>> rtp.valid  # Padding enabled but not present
False
```

Testing header fields value:

```
>>> bytes = bytes.fromhex('80 a1 a4 25 ca fe b5 04 b0 60 5e bb 12 34')
>>> rtp = RtpPacket(bytes, len(bytes))
>>> rtp.valid
True
>>> print(rtp)
version      = 2
errors       = []
padding      = False
extension    = False
marker       = True
payload type = 33
sequence     = 42021
timestamp    = 3405690116
clock rate   = 90000
time         = 37841
ssrc         = 2959105723
csrc count   = 0
payload size = 2
>>> rtp.csrc
[]
>>> rtp.payload[0]
18
>>> rtp.payload[1]
52
```

Testing header fields value (with padding, extension and ccrc):

```
>>> bytes = bytes.fromhex('b5a1a401 cafea421 b0605ebb 11111111 22222222␣
↪33333333 '
...                        '44444444 55555555 00000004 87654321 12340002')
>>> rtp = RtpPacket(bytes, len(bytes))
>>> rtp.valid
True
>>> rtp.version
2
>>> rtp.padding
True
>>> rtp.extension
True
>>> rtp.marker
True
>>> rtp.payload_type
33
>>> rtp.sequence
41985
>>> rtp.timestamp
3405685793
>>> rtp.clock_rate
90000
>>> rtp.ssrc
2959105723
>>> len(rtp.csrc)
5
>>> rtp.csrc
[286331153, 572662306, 858993459, 1145324612, 1431655765]
>>> rtp.payload
bytearray(b'\x124')
```

classmethod create(*sequence*, *timestamp*, *payload_type*, *payload*)
Create a valid RTP packet with a given payload.

Parameters

- **sequence** (*int*) – Packet sequence number (16 bits)
- **timestamp** (*int*) – Packet timestamp (32 bits)
- **payload_type** (*int*) – Packet payload type (7 bits)
- **payload** (*bytearray*) – Packet payload, can be an array of bytes or a string

Example usage

```
>>> p = RtpPacket.create(10, 1024, RtpPacket.MP2T_PT, 'The payload string')
>>> q = RtpPacket.create(11, 1028, RtpPacket.MP2T_PT, bytearray.fromhex('00␣
↪11 22 33'))
>>> r = RtpPacket.create(11, 1028, RtpPacket.DYNAMIC_PT, bytearray.fromhex(
↪'cc aa ff ee'))
>>> assert p.validMP2T and q.validMP2T and r.valid
```

pytoolbox.network.url module

pytoolbox.network.url.**with_subdomain**(*url*, *subdomain=None*)
Return the `url` with the sub-domain replaced with `subdomain`.

Example usage

```
>>> from pytoolbox.unittest import asserts
>>> eq = asserts.equal
>>> asserts.equal(
...     with_subdomain('http://app.website.com/page'),
...     'http://website.com/page')
>>> asserts.equal(
...     with_subdomain('http://app.website.com/page', 'help'),
...     'http://help.website.com/page')
>>> asserts.equal(
...     with_subdomain('https://app.website.com#d?page=1', 'help'),
...     'https://help.website.com#d?page=1')
```

pytoolbox.rest_framework package

Subpackages

pytoolbox.rest_framework.metadata package

Submodules

pytoolbox.rest_framework.metadata.mixins module

Mix-ins for building your own Django REST Framework powered API metadata .

class pytoolbox.rest_framework.metadata.mixins.**ExcludeRelatedChoicesMixin**
Bases: object

Do not includes related fields to avoid having choices with hundreds instances.

related_fields = (<class 'rest_framework.relations.RelatedField'>, <class 'rest_framew

get_field_info(*field*)

pytoolbox.rest_framework.serializers package

Submodules

pytoolbox.rest_framework.serializers.fields module

Extra fields for building your own Django REST Framework powered API serializers.

class pytoolbox.rest_framework.serializers.fields.**StripCharField**(***kwargs*)
Bases: rest_framework.fields.CharField

__init__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

to_internal_value(*data*)
Transform the *incoming* primitive data into a native value.

pytoolbox.rest_framework.serializers.mixins module

Mix-ins for building your own Django REST Framework powered API serializers.

class pytoolbox.rest_framework.serializers.mixins.**BaseModelMixin**
Bases: object

build_url_field(*field_name*, *model_class*)

class pytoolbox.rest_framework.serializers.mixins.**FromPrivateKeyMixin**
Bases: object

Allow to provide the PK of the model to retrieve it instead of creating a new instance with fields from data.

default_error_messages

to_internal_value(*data*)
Transform the *incoming* primitive data into a native value.

create(*validated_data*)

class pytoolbox.rest_framework.serializers.mixins.**NestedWriteMixin**
Bases: object

to_internal_value(*data*)
Return a tuple with (self, validate_data) to allow working on validated data with this serializer.

class pytoolbox.rest_framework.serializers.mixins.**ReadOnlyMixin**(**args*, ***kwargs*)
Bases: object

__init__(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

create(*validated_data*)

update(*task*, *validated_data*)

pytoolbox.rest_framework.views package

Submodules

pytoolbox.rest_framework.views.mixins module

Submodules

pytoolbox.rest_framework.permissions module

pytoolbox.selenium package

class pytoolbox.selenium.**Keys**
Bases: object

Set of special keys codes.

NULL = '\ue000'

CANCEL = '\ue001'

HELP = '\ue002'

BACKSPACE = '\ue003'

BACK_SPACE = '\ue003'

TAB = '\ue004'

CLEAR = '\ue005'

RETURN = '\ue006'

ENTER = '\ue007'

SHIFT = '\ue008'

LEFT_SHIFT = '\ue008'

CONTROL = '\ue009'

LEFT_CONTROL = '\ue009'

ALT = '\ue00a'

LEFT_ALT = '\ue00a'

PAUSE = '\ue00b'

ESCAPE = '\ue00c'

SPACE = '\ue00d'

PAGE_UP = '\ue00e'

PAGE_DOWN = '\ue00f'

END = '\ue010'

HOME = '\ue011'

LEFT = '\ue012'

ARROW_LEFT = '\ue012'

```
    UP = '\ue013'
    ARROW_UP = '\ue013'
    RIGHT = '\ue014'
    ARROW_RIGHT = '\ue014'
    DOWN = '\ue015'
    ARROW_DOWN = '\ue015'
    INSERT = '\ue016'
    DELETE = '\ue017'
    SEMICOLON = '\ue018'
    EQUALS = '\ue019'
    NUMPAD0 = '\ue01a'
    NUMPAD1 = '\ue01b'
    NUMPAD2 = '\ue01c'
    NUMPAD3 = '\ue01d'
    NUMPAD4 = '\ue01e'
    NUMPAD5 = '\ue01f'
    NUMPAD6 = '\ue020'
    NUMPAD7 = '\ue021'
    NUMPAD8 = '\ue022'
    NUMPAD9 = '\ue023'
    MULTIPLY = '\ue024'
    ADD = '\ue025'
    SEPARATOR = '\ue026'
    SUBTRACT = '\ue027'
    DECIMAL = '\ue028'
    DIVIDE = '\ue029'
    F1 = '\ue031'
    F2 = '\ue032'
    F3 = '\ue033'
    F4 = '\ue034'
    F5 = '\ue035'
    F6 = '\ue036'
    F7 = '\ue037'
    F8 = '\ue038'
    F9 = '\ue039'
    F10 = '\ue03a'
```

```
F11 = '\ue03b'
F12 = '\ue03c'
META = '\ue03d'
COMMAND = '\ue03d'
ZENKAKU_HANKAKU = '\ue040'
```

Subpackages

pytoolbox.selenium.webelements package

Submodules

pytoolbox.selenium.webelements.base module

pytoolbox.selenium.webelements.bootstrap_slider module

pytoolbox.selenium.webelements.bootstrap_switch module

Submodules

pytoolbox.selenium.client module

pytoolbox.selenium.common module

class pytoolbox.selenium.common.**FindMixin**

Bases: object

static clean_elements(*elements*, *criteria*, *force_list=False*, *fail=True*)

find_css(*css_selector*, *prefix=True*, *force_list=False*, *fail=True*)

find_id(*element_id*, *prefix=True*, *force_list=False*, *fail=True*)

find_name(*element_name*, *prefix=True*, *force_list=False*, *fail=True*)

find_xpath(*xpath*, *force_list=False*, *fail=True*)

pytoolbox.selenium.exceptions module

exception pytoolbox.selenium.exceptions.**NoSuchSpecializedElementException**(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.NoSuchElementException

exception pytoolbox.selenium.exceptions.**ElementClickInterceptedException**(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

The Element Click command could not be completed because the element receiving the events is obscuring the element that was requested to be clicked.

exception pytoolbox.selenium.exceptions.**ElementNotInteractableException**(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.InvalidElementStateException

Thrown when an element is present in the DOM but interactions with that element will hit another element due to paint order.

exception pytoolbox.selenium.exceptions.**ElementNotSelectableException**(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.InvalidElementStateException

Thrown when trying to select an unselectable element.

For example, selecting a 'script' element.

exception pytoolbox.selenium.exceptions.**ElementNotVisibleException**(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.InvalidElementStateException

Thrown when an element is present on the DOM, but it is not visible, and so is not able to be interacted with.

Most commonly encountered when trying to click or read text of an element that is hidden from view.

exception pytoolbox.selenium.exceptions.**ImeActivationFailedException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when activating an IME engine has failed.

exception pytoolbox.selenium.exceptions.**ImeNotAvailableException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when IME support is not available.

This exception is thrown for every IME-related method call if IME support is not available on the machine.

exception pytoolbox.selenium.exceptions.**InsecureCertificateException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Navigation caused the user agent to hit a certificate warning, which is usually the result of an expired or invalid TLS certificate.

exception pytoolbox.selenium.exceptions.**InvalidArgumentException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

The arguments passed to a command are either invalid or malformed.

exception pytoolbox.selenium.exceptions.**InvalidCookieDomainException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when attempting to add a cookie under a different domain than the current URL.

exception pytoolbox.selenium.exceptions.**InvalidCoordinatesException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

The coordinates provided to an interaction's operation are invalid.

exception pytoolbox.selenium.exceptions.**InvalidElementStateException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when a command could not be completed because the element is in an invalid state.

This can be caused by attempting to clear an element that isn't both editable and resettable.

exception pytoolbox.selenium.exceptions.**InvalidSelectorException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when the selector which is used to find an element does not return a WebElement.

Currently this only happens when the selector is an xpath expression and it is either syntactically invalid (i.e. it is not a xpath expression) or the expression does not select WebElements (e.g. "count(//input)").

__init__(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*) → None

Initialize self. See help(type(self)) for accurate signature.

exception pytoolbox.selenium.exceptions.**InvalidSessionIdException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Occurs if the given session id is not in the list of active sessions, meaning the session either does not exist or that it's not active.

exception pytoolbox.selenium.exceptions.**InvalidSwitchToTargetException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when frame or window target to be switched doesn't exist.

exception pytoolbox.selenium.exceptions.**JavascriptException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

An error occurred while executing JavaScript supplied by the user.

exception pytoolbox.selenium.exceptions.**MoveTargetOutOfBoundsException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when the target provided to the *ActionsChains* move() method is invalid, i.e. out of document.

exception pytoolbox.selenium.exceptions.**NoAlertPresentException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when switching to no presented alert.

This can be caused by calling an operation on the Alert() class when an alert is not yet on the screen.

exception pytoolbox.selenium.exceptions.**NoSuchAttributeException**(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when the attribute of element could not be found.

You may want to check if the attribute exists in the particular browser you are testing against. Some browsers may have different property names for the same property. (IE8's .innerText vs. Firefox .textContent)

exception pytoolbox.selenium.exceptions.**NoSuchCookieException**(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

No cookie matching the given path name was found amongst the associated cookies of the current browsing context's active document.

exception pytoolbox.selenium.exceptions.**NoSuchDriverException**(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Raised when driver is not specified and cannot be located.

__init__(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*) → None

Initialize self. See help(type(self)) for accurate signature.

exception pytoolbox.selenium.exceptions.**NoSuchElementException**(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when element could not be found.

If you encounter this exception, you may want to check the following:

- Check your selector used in your find_by...
- Element may not yet be on the screen at the time of the find operation, (webpage is still loading) see selenium.webdriver.support.wait.WebDriverWait() for how to write a wait wrapper to wait for an element to appear.

__init__(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*) → None
Initialize self. See help(type(self)) for accurate signature.

exception pytoolbox.selenium.exceptions.**NoSuchFrameException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.InvalidSwitchToTargetException

Thrown when frame target to be switched doesn't exist.

exception pytoolbox.selenium.exceptions.**NoSuchShadowRootException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when trying to access the shadow root of an element when it does not have a shadow root attached.

exception pytoolbox.selenium.exceptions.**NoSuchWindowException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.InvalidSwitchToTargetException

Thrown when window target to be switched doesn't exist.

To find the current set of active window handles, you can get a list of the active window handles in the following way:

```
print driver.window_handles
```

exception pytoolbox.selenium.exceptions.**ScreenshotException**(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

A screen capture was made impossible.

exception pytoolbox.selenium.exceptions.**SessionNotCreatedException**(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

A new session could not be created.

exception pytoolbox.selenium.exceptions.**StaleElementReferenceException**(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when a reference to an element is now "stale".

Stale means the element no longer appears on the DOM of the page.

Possible causes of StaleElementReferenceException include, but not limited to:

- You are no longer on the same page, or the page may have refreshed since the element was located.
- The element may have been removed and re-added to the screen, since it was located. Such as an element being relocated. This can happen typically with a javascript framework when values are updated and the node is rebuilt.
- Element may have been inside an iframe or another context which was refreshed.

__init__(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*) → None

Initialize self. See help(type(self)) for accurate signature.

exception pytoolbox.selenium.exceptions.**TimeoutException**(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when a command does not complete in enough time.

exception pytoolbox.selenium.exceptions.**UnableToSetCookieException**(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when a driver fails to set a cookie.

exception pytoolbox.selenium.exceptions.**UnexpectedAlertPresentException**(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*, *alert_text: Optional[str] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when an unexpected alert has appeared.

Usually raised when an unexpected modal is blocking the webdriver from executing commands.

__init__(*msg: Optional[str] = None*, *screen: Optional[str] = None*, *stacktrace: Optional[Sequence[str]] = None*, *alert_text: Optional[str] = None*) → None

Initialize self. See help(type(self)) for accurate signature.

exception pytoolbox.selenium.exceptions.**UnexpectedTagNameException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

Thrown when a support class did not get an expected web element.

exception pytoolbox.selenium.exceptions.**UnknownMethodException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: selenium.common.exceptions.WebDriverException

The requested command matched a known URL but did not match any methods for that URL.

exception pytoolbox.selenium.exceptions.**WebDriverException**(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*)

Bases: Exception

Base webdriver exception.

__init__(*msg: Optional[str] = None, screen: Optional[str] = None, stacktrace: Optional[Sequence[str]] = None*) → None

Initialize self. See help(type(self)) for accurate signature.

pytoolbox.selenium.select module

class pytoolbox.selenium.select.**Select**(*webelement: selenium.webdriver.remote.webelement.WebElement*)

Bases: selenium.webdriver.support.select.Select

A Select with the attributes of the WebElement.

pytoolbox.selenium.test module

pytoolbox.selenium.webdrivers module

1.1.2 Submodules

pytoolbox.argparse module

pytoolbox.atlassian module

Module related to managing projects with Atlassian's products.

class pytoolbox.atlassian.**JiraProject**(*project=None*, *server=None*, *auth=None*, *feature_type=None*)

Bases: object

A JIRA project class with a simple API leveraging the class `jira.JIRA`.

__init__(*project=None*, *server=None*, *auth=None*, *feature_type=None*)
Initialize self. See help(type(self)) for accurate signature.

fields

features

jira

versions

get_field(*name*, *fail=True*)

get_field_value(*issue*, *name*, *default=None*)

pytoolbox.collections module

class pytoolbox.collections.**EventsTable**(*sparse_events_table*, *time_range*, *time_speedup*, *sleep_factor=1.0*)

Bases: object

Scan a spare events table and replace missing entry by previous (non empty) entry.

__init__(*sparse_events_table*, *time_range*, *time_speedup*, *sleep_factor=1.0*)
Initialize self. See help(type(self)) for accurate signature.

get(*time*, *time_speedup=None*, *default_value=None*)

sleep_time(*time*, *time_speedup=None*, *sleep_factor=None*)
Return required sleep time to wait for next scheduled event.

Example usage

```
>>> table = EventsTable({0: 'salut'}, 24, 60)
>>> table.sleep_time(1)
59
>>> table.sleep_time(58)
2
>>> table.sleep_time(60)
60
>>> table.sleep_time(62)
58
>>> table.sleep_time(3590, time_speedup=1)
10
>>> table.sleep_time(12543, time_speedup=1)
1857
>>> table.sleep_time(12543, time_speedup=1, sleep_factor=2)
57
>>> table.sleep_time(12600, time_speedup=1, sleep_factor=2)
```

(continues on next page)

(continued from previous page)

```
1800
>>> table.sleep_time(1, time_speedup=60, sleep_factor=1)
59
>>> table.sleep_time(1, time_speedup=60, sleep_factor=2)
29
>>> table.sleep_time(30, time_speedup=60, sleep_factor=2)
30
>>> table.time_range = 1
>>> table.sleep_time(1, time_speedup=1)
149
```

class pytoolbox.collections.**pygal_deque**

Bases: collections.deque

A deque None'ing duplicated values to produce nicer pygal line charts (e.g. 5555322211111 -> 5__532_21___1).

Warning: Not yet implemented:

- appendleft(x)
- clear()
- count(x)
- extend(iterable)
- extendleft(iterable)
- pop()
- popleft()
- remove(value)
- reverse()
- rotate(n)

last = None

append(*value*)

Add an element to the right side of the deque.

list(*fill=False*)

pytoolbox.collections.**flatten_dict**(*the_dict*, *key_template='{0}.{1}'*)

Flatten the keys of a nested dictionary. Nested keys will be appended iteratively using given *key_template*.

Example usage

```
>>> flatten_dict({'a': 'b', 'c': 'd'})
{'a': 'b', 'c': 'd'}
>>> flatten_dict({'a': {'b': {'c': ['d', 'e']}, 'f': 'g'}})
{'a.b.c': ['d', 'e'], 'a.f': 'g'}
>>> flatten_dict({'a': {'b': {'c': ['d', 'e']}, 'f': 'g'}}, '{1}-{0}')
{'c-b-a': ['d', 'e'], 'f-a': 'g'}
```

pytoolbox.collections.**merge_dicts**(**dicts*)

Return a dictionary from multiple dictionaries.

Warning: This operation is not commutative.

Example usage

```
>>> merge_dicts({'a': 1, 'b': 2}, {'b': 3, 'c': 4}, {'c': 5})
{'a': 1, 'b': 3, 'c': 5}
>>> merge_dicts({'c': 5}, {'b': 3, 'c': 4}, {'a': 1, 'b': 2})
{'c': 4, 'b': 2, 'a': 1}
```

pytoolbox.collections.**swap_dict_of_values**(*the_dict*, *type=<class 'set'>*, *method=<method 'add' of 'set' objects>*)

Return a dictionary (collections.defaultdict) with keys and values swapped.

This algorithm expect that the values are a container with objects, not a single object. Set type to None if values are unique and you want keys to be the values.

Example usage

Simple swap:

```
>>> swap_dict_of_values({'odd': [1, 3], 'even': (0, 2)}, type=None)
{1: 'odd', 3: 'odd', 0: 'even', 2: 'even'}
```

Complex swap:

```
>>> def S(value):
...     return {k: sorted(v) for k, v in sorted(value.items())}
...
>>> S(swap_dict_of_values(
...     {'odd': [1, 3], 'even': (0, 2), 'fib': {1, 2, 3}},
...     type=list,
...     method=list.append))
{0: ['even'], 1: ['fib', 'odd'], 2: ['even', 'fib'], 3: ['fib', 'odd']}
>>> swap_dict_of_values({'o': [1, 3], 'e': (0, 2), 'f': {2, 3}}, method='add')[2]␣
↪== {'f', 'e'}
True
>>> swap_dict_of_values({'bad': 'ab', 'example': 'ab'})['a'] == {'bad', 'example'}
True
```

pytoolbox.collections.**to_dict_of_values**(*iterable*, *type=<class 'list'>*, *method=<method 'append' of 'list' objects>*)

Return a dictionary (collections.defaultdict) with key, value pairs merged as key -> values.

Example usage

```
>>> from pytoolbox.unittest import asserts
>>> asserts.dict_equal(
...     to_dict_of_values([('odd', 1), ('odd', 3), ('even', 0), ('even', 2)]),
...     {'even': [0, 2], 'odd': [1, 3]})
>>> asserts.dict_equal(
...     to_dict_of_values((('a', 1), ('a', 1), ('a', 2)), type=set, method=set.
↪add),
...     {'a': {1, 2}})
```

pytoolbox.collections.**window**(*values*, *index*, *delta*)

Extract 1+2*‘delta‘ items from *values* centered at *index* and return a tuple with (items, left, right).

This function tries to simulate a physical slider, meaning the number of extracted elements is constant but the centering at *index* is not guaranteed.

A visual example with 6 *values* and `delta=1`:

```
index = 0  [+++]---  left = 0, right = 2
index = 1  [+++]---  left = 0, right = 2
index = 2  -[+++]--  left = 1, right = 3
index = 3  --[+++]-  left = 2, right = 4
index = 4  ---[+++]  left = 3, right = 5
index = 5  ---[+++]  left = 3, right = 5
```

Example usage

```
>>> window(['a'], 0, 2)
(['a'], 0, 0)
>>> window(['a', 'b', 'c', 'd'], 2, 0)
(['c'], 2, 2)
>>> window(['a', 'b', 'c', 'd', 'e'], 0, 1)
(['a', 'b', 'c'], 0, 2)
>>> window(['a', 'b', 'c', 'd', 'e'], 1, 1)
(['a', 'b', 'c'], 0, 2)
>>> window(['a', 'b', 'c', 'd', 'e'], 2, 1)
(['b', 'c', 'd'], 1, 3)
>>> window(['a', 'b', 'c', 'd', 'e'], 3, 1)
(['c', 'd', 'e'], 2, 4)
>>> window(['a', 'b', 'c', 'd', 'e'], 4, 1)
(['c', 'd', 'e'], 2, 4)
>>> window(['a', 'b', 'c', 'd', 'e'], 3, 6)
(['a', 'b', 'c', 'd', 'e'], 0, 4)
>>> data = list(range(20))
>>> window(data, 6, 6)
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], 0, 12)
>>> window(data, 7, 6)
([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13], 1, 13)
>>> window(data, 10, 6)
([4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16], 4, 16)
>>> window(data, 19, 6)
([7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19], 7, 19)
```

pytoolbox.comparison module

class pytoolbox.comparison.**SlotsEqualityMixin**

Bases: object

Implement the comparison operators based on the slots. Both the name of the slots retrieved with *pytoolbox.types.get_slots()* and theirs values are tested for equality.

pytoolbox.comparison.**unified_diff**(*before: str*, *after: str*, *, *colorize: bool = True*, ***kwargs*) → str

Colorization is not guaranteed (your environment may disable it). Use *pytoolbox.console.toggle_colors* appropriately to ensure it.

pytoolbox.comparison.**compare_versions**(*a: str*, *b: str*, *operator: str*) → bool | None

pytoolbox.comparison.**satisfy_version_constraints**(*version: str | None*, *constraints: tuple[str, ...]*, *, *default='<undefined>'*) → bool

Ensure given version fulfill the constraints (if any).

Constraints are given in the form '<operator> <version>', Exemple:

```
>>> satisfy_version_constraints('v1.5.2', ['>= v1.5', '< v2'])
True
>>> satisfy_version_constraints('v0.7', ['>= v1.5', '< v2'])
False
>>> satisfy_version_constraints(None, ['>= v1.5', '< v2'])
False
>>> satisfy_version_constraints('main', ['!= main'])
False
>>> satisfy_version_constraints(None, ['== <undefined>'])
True
>>> satisfy_version_constraints(None, ['!= master'], default='master')
False
```

class pytoolbox.comparison.**Version**(*version: str*)

Bases: packaging.version._BaseVersion

This class abstracts handling of a project's versions.

A *Version* instance is comparison aware and can be compared and sorted using the standard Python interfaces.

```
>>> v1 = Version("1.0a5")
>>> v2 = Version("1.0")
>>> v1
<Version('1.0a5')>
>>> v2
<Version('1.0')>
>>> v1 < v2
True
>>> v1 == v2
False
>>> v1 > v2
False
>>> v1 >= v2
False
>>> v1 <= v2
True
```

__init__(*version: str*) → None

Initialize a Version object.

Parameters **version** – The string representation of a version which will be parsed and normalized before use.

Raises **InvalidVersion** – If the version does not conform to PEP 440 in any way then this exception will be raised.

epoch

The epoch of the version.

```
>>> Version("2.0.0").epoch
0
>>> Version("1!2.0.0").epoch
1
```

release

The components of the "release" segment of the version.

```
>>> Version("1.2.3").release
(1, 2, 3)
```

(continues on next page)

(continued from previous page)

```
>>> Version("2.0.0").release
(2, 0, 0)
>>> Version("1!2.0.0.post0").release
(2, 0, 0)
```

Includes trailing zeroes but not the epoch or any pre-release / development / post-release suffixes.

pre

The pre-release segment of the version.

```
>>> print(Version("1.2.3").pre)
None
>>> Version("1.2.3a1").pre
('a', 1)
>>> Version("1.2.3b1").pre
('b', 1)
>>> Version("1.2.3rc1").pre
('rc', 1)
```

post

The post-release number of the version.

```
>>> print(Version("1.2.3").post)
None
>>> Version("1.2.3.post1").post
1
```

dev

The development number of the version.

```
>>> print(Version("1.2.3").dev)
None
>>> Version("1.2.3.dev1").dev
1
```

local

The local version segment of the version.

```
>>> print(Version("1.2.3").local)
None
>>> Version("1.2.3+abc").local
'abc'
```

public

The public portion of the version.

```
>>> Version("1.2.3").public
'1.2.3'
>>> Version("1.2.3+abc").public
'1.2.3'
>>> Version("1.2.3+abc.dev1").public
'1.2.3'
```

base_version

The “base version” of the version.

```
>>> Version("1.2.3").base_version
'1.2.3'
>>> Version("1.2.3+abc").base_version
'1.2.3'
>>> Version("1!1.2.3+abc.dev1").base_version
'1!1.2.3'
```

The "base version" is the public version of the project without any pre or post release markers.

is_prerelease
Whether this version is a pre-release.

```
>>> Version("1.2.3").is_prerelease
False
>>> Version("1.2.3a1").is_prerelease
True
>>> Version("1.2.3b1").is_prerelease
True
>>> Version("1.2.3rc1").is_prerelease
True
>>> Version("1.2.3dev1").is_prerelease
True
```

is_postrelease
Whether this version is a post-release.

```
>>> Version("1.2.3").is_postrelease
False
>>> Version("1.2.3.post1").is_postrelease
True
```

is_devrelease
Whether this version is a development release.

```
>>> Version("1.2.3").is_devrelease
False
>>> Version("1.2.3.dev1").is_devrelease
True
```

major
The first item of *release* or `0` if unavailable.

```
>>> Version("1.2.3").major
1
```

minor
The second item of *release* or `0` if unavailable.

```
>>> Version("1.2.3").minor
2
>>> Version("1").minor
0
```

micro
The third item of *release* or `0` if unavailable.

```
>>> Version("1.2.3").micro
3
>>> Version("1").micro
0
```

pytoolbox.comparison.**try_parse_version**(*version: str*) → str | Version

pytoolbox.console module

pytoolbox.crypto module

pytoolbox.datetime module

pytoolbox.datetime.**datetime_now**(*format='%Y-%m-%d %H:%M:%S'*, *append_utc=False*, *offset=None*, *tz=<UTC>*)

Return the current (timezone aware) date and time as UTC, local (tz=None) or related to a timezone. If *format* is not None, the date will be returned in a formatted string.

Parameters

- **format** (*str*) – Output date string formatting
- **append_utc** (*bool*) – Append ' UTC' to date string
- **offset** (*datetime.timedelta*) – Offset to add to current time
- **tz** (*tz*) – The timezone (e.g. `pytz.timezone('EST')`)

Example usage

Add an offset:

```
>>> now = datetime_now(format=None)
>>> future = datetime_now(offset=datetime.timedelta(hours=2, minutes=10),␣
↪format=None)
>>> result = (future - now)
>>> type(result)
<class 'datetime.timedelta'>
>>> print(result)
2:10:00.00...
```

Append UTC to output date string:

```
>>> type(datetime_now())
<class 'str'>
>>> assert ' UTC' not in datetime_now(tz=pytz.utc, append_utc=False)
>>> assert ' UTC' not in datetime_now(tz=None, append_utc=True)
>>> assert ' UTC' not in datetime_now(tz=pytz.timezone('EST'), append_utc=True)
>>> assert ' UTC' in datetime_now(tz=pytz.utc, append_utc=True)
```

Play with timezones:

```
>> datetime_now(tz=pytz.timezone('Europe/Zurich'))
'2013-10-17 09:54:08'
>> datetime_now(tz=pytz.timezone('US/Eastern'))
'2013-10-17 03:54:08'
```

pytoolbox.datetime.**datetime_to_str**(*date_time*, *format='%Y-%m-%d %H:%M:%S'*, *append_utc=False*)

pytoolbox.datetime.**str_to_datetime**(*date*, *format='%Y-%m-%d %H:%M:%S'*, *fail=True*)

Return the *date* string converted into an instance of datetime.datetime. Handle 24h+ hour format like 2015:06:28 24:05:00 equal to the 28th June 2015 at midnight and 5 minutes.

Example usage

```
>>> str_to_datetime('1985-01-06 05:02:00')
datetime.datetime(1985, 1, 6, 5, 2)
>>> str_to_datetime('this is not a date')
Traceback (most recent call last):
    ...
ValueError: time data 'this is not a date' does not match format '%Y-%m-%d %H:%M:
↪%S'
```

pytoolbox.datetime.**multiply_time**(*value*, *factor*, *as_delta=False*)

Return an instance of datetime.time/datetime.timedelta corresponding to *value* multiplied by a *factor*.

Example usage

```
>>> multiply_time('00:10:00', 0.5)
datetime.time(0, 5)
>>> multiply_time(datetime.timedelta(seconds=60), 3)
datetime.time(0, 3)
>>> multiply_time(120, 0.1)
datetime.time(0, 0, 12)
>>> res = multiply_time(datetime.timedelta(seconds=152, microseconds=500000), 1,␣
↪as_delta=True)
>>> type(res)
<class 'datetime.timedelta'>
>>> print(res)
0:02:32.500000
```

pytoolbox.datetime.**parts_to_time**(*hours*, *minutes*, *seconds*, *microseconds*, *as_delta=False*)

Return an instance of datetime.time/datetime.timedelta out of the parts.

Example usage

```
>>> parts_to_time(23, 15, 7, 3500)
datetime.time(23, 15, 7, 3500)
>>> result = parts_to_time(23, 15, 7, 3500, as_delta=True)
>>> type(result)
<class 'datetime.timedelta'>
>>> print(result)
23:15:07.003500
```

pytoolbox.datetime.**secs_to_time**(*value*, *defaults_to_zero=False*, *as_delta=False*)

Return an instance of datetime.time/datetime.timedelta, taking *value* as the number of seconds + microseconds (e.g. 10.3 = 10s 3000us).

Example usage

```
>>> secs_to_time(83707.0035)
datetime.time(23, 15, 7, 3500)
>>> secs_to_time(None)
>>> secs_to_time(None, defaults_to_zero=True)
datetime.time(0, 0)
>>> result = secs_to_time(83707.0035, as_delta=True)
>>> type(result)
```

(continues on next page)

(continued from previous page)

```
<class 'datetime.timedelta'>
>>> print(result)
23:15:07.003500
>>> secs_to_time(None, as_delta=True) is None
True
>>> secs_to_time(None, defaults_to_zero=True, as_delta=True)
datetime.timedelta(0)
```

pytoolbox.datetime.**str_to_time**(*value*, *defaults_to_zero=False*, *as_delta=False*)

Return the string of format 'hh:mm:ss' into an instance of time.

Example usage

```
>>> str_to_time('08:23:57')
datetime.time(8, 23, 57)
>>> str_to_time('00:03:02.12')
datetime.time(0, 3, 2, 120)
>>> str_to_time(None)
>>> str_to_time(None, defaults_to_zero=True)
datetime.time(0, 0)
>>> result = str_to_time('08:23:57', as_delta=True)
>>> type(result)
<class 'datetime.timedelta'>
>>> str(result)
'8:23:57'
>>> result = str_to_time('00:03:02.12', as_delta=True)
>>> type(result)
<class 'datetime.timedelta'>
>>> str(result)
'0:03:02.120000'
>>> str_to_time(None, as_delta=True) is None
True
>>> str_to_time(None, defaults_to_zero=True, as_delta=True)
datetime.timedelta(0)
```

pytoolbox.datetime.**time_ratio**(*numerator*, *denominator*, *zero_div_result=1.0*)

Return the ratio between two times.

Example usage

```
>>> from pytoolbox.unittest import asserts
>>> time_ratio('0:30:00', '01:30:00')
0.33...
>>> time_ratio('0:00:05', '00:00:00')
1.0
>>> with asserts.raises(ValueError):
...     time_ratio('01:42:34', 'N/A')
```

pytoolbox.datetime.**total_seconds**(*time*)

Return the *time* converted in seconds.

Example usage

```
>>> total_seconds('00:10:00')
600.0
>>> total_seconds('01:54:17')
6857.0
```

(continues on next page)

(continued from previous page)

```
>>> round(total_seconds('16.40'), 3)
16.4
>>> total_seconds(143.2)
143.2
>>> total_seconds(datetime.timedelta(seconds=152, microseconds=500000))
152.5
>>> total_seconds(datetime.datetime(2010, 6, 10, 0, 1, 30))
90.0
>>> total_seconds(datetime.datetime(2010, 6, 10, 14, 15, 23))
51323.0
>>> total_seconds(datetime.datetime(2010, 6, 10, 23, 59, 59))
86399.0
```

pytoolbox.datetime.**datetime_to_epoch**(*date_time*, *utc=True*, *factor=1*)

Return the `datetime.datetime`/`datetime.date` converted into an Unix epoch. Default *factor* means that the result is in seconds.

Example usage

```
>>> datetime_to_epoch(datetime.datetime(1970, 1, 1), factor=1)
0
>>> datetime_to_epoch(datetime.datetime(2010, 6, 10))
1276128000
>>> datetime_to_epoch(datetime.datetime(2010, 6, 10), factor=1000)
1276128000000
>>> datetime_to_epoch(datetime.date(2010, 6, 10), factor=1000)
1276128000000
>>> datetime_to_epoch(datetime.date(1987, 6, 10), factor=1000)
550281600000
```

In Switzerland:

```
>> datetime_to_epoch(datetime.datetime(1970, 1, 1), utc=False, factor=1)
-3600
>> datetime_to_epoch(datetime.date(1970, 1, 1), utc=False, factor=1)
-3600
```

pytoolbox.datetime.**epoch_to_datetime**(*unix_epoch*, *tz=<UTC>*, *factor=1*)

Return the Unix epoch converted to a `datetime.datetime`. Default *factor* means that the *unix_epoch* is in seconds.

Example usage

```
>>> epoch_to_datetime(0, factor=1)
datetime.datetime(1970, 1, 1, 0, 0, tzinfo=<UTC>)
>>> epoch_to_datetime(1276128000, factor=1)
datetime.datetime(2010, 6, 10, 0, 0, tzinfo=<UTC>)
>>> epoch_to_datetime(1276128000, tz=pytz.timezone('Europe/Zurich'), factor=1)
datetime.datetime(2010, 6, 10, 2, 0, tzinfo=<DstTzInfo 'Europe/Zurich'␣
↪CEST+2:00:00 DST>)
>>> epoch_to_datetime(1276128000000, factor=1000)
datetime.datetime(2010, 6, 10, 0, 0, tzinfo=<UTC>)
>>> today = datetime.datetime(1985, 6, 1, 5, 2, 0, tzinfo=pytz.utc)
>>> epoch_to_datetime(datetime_to_epoch(today, factor=1000), factor=1000) == today
True
```

pytoolbox.decorators module

pytoolbox.enum module

Module related to enumeration.

class pytoolbox.enum.**OrderedEnum**
Bases: enum.Enum

An enumeration both hash-able and ordered by value.

Reference: https://docs.python.org/3/library/enum.html#orderedenum.

pytoolbox.exceptions module

exception pytoolbox.exceptions.**MessageMixin**(*message=None*, ***kwargs*)
Bases: Exception

__init__(*message=None*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

message = None

exception pytoolbox.exceptions.**BadHTTPResponseCodeError**(*message=None*, ***kwargs*)
Bases: *pytoolbox.exceptions.MessageMixin*, Exception

message = 'Download request {url} code {r_code} expected {code}.'

exception pytoolbox.exceptions.**CorruptedFileError**(*message=None*, ***kwargs*)
Bases: *pytoolbox.exceptions.MessageMixin*, Exception

message = 'File {path} is corrupted checksum {file_hash} expected {expected_hash}.'

exception pytoolbox.exceptions.**ForbiddenError**
Bases: Exception

A forbidden error.

exception pytoolbox.exceptions.**InvalidBrandError**(*message=None*, ***kwargs*)
Bases: *pytoolbox.exceptions.MessageMixin*, Exception

message = 'Brand {brand} not in {brands}.'

exception pytoolbox.exceptions.**InvalidIPSocketError**(*message=None*, ***kwargs*)
Bases: *pytoolbox.exceptions.MessageMixin*, Exception

message = '{socket} is not a valid IP socket.'

exception pytoolbox.exceptions.**MultipleSignalHandlersError**(*message=None*, ***kwargs*)
Bases: *pytoolbox.exceptions.MessageMixin*, Exception

message = 'Signal {signum} already handled by {handlers}.'

exception pytoolbox.exceptions.**UndefinedPathError**
Bases: Exception

pytoolbox.exceptions.**assert_raises_item**(*exception_cls*, *something*, *index*, *value=None*, *delete=False*)

Example usage

```
>>> x = {0: 3.14, 1: 2.54}
```

Assert that __getitem__ will fail:

```
>>> assert_raises_item(KeyError, x, 2)
>>> assert_raises_item(ValueError, x, 3)
Traceback (most recent call last):
    ...
ValueError: Exception KeyError is not an instance of ValueError.
>>> assert_raises_item(Exception, x, 0)
Traceback (most recent call last):
    ...
AssertionError: Exception Exception not raised.
```

Assert that __setitem__ will fail:

```
>>> assert_raises_item(TypeError, x, [10], value=3.1415)
>>> assert_raises_item(TypeError, x, 0, value=3.1415)
Traceback (most recent call last):
    ...
AssertionError: Exception TypeError not raised.
```

Assert that __delitem__ will fail:

```
>>> assert_raises_item(KeyError, x, 2, delete=True)
>>> assert_raises_item(KeyError, x, 1, delete=True)
Traceback (most recent call last):
    ...
AssertionError: Exception KeyError not raised.
```

```
>>> x == {0: 3.1415}
True
```

pytoolbox.exceptions.**get_exception_with_traceback**(*exception*)

Return a string with the exception traceback.

Example usage

If the exception was not raised then there are no traceback:

```
>>> get_exception_with_traceback(ValueError('yé'))
'ValueError: yé\n'
```

If the exception was raised then there is a traceback:

```
>>> try:
...     raise RuntimeError('yé')
... except Exception as ex:
...     trace = get_exception_with_traceback(ex)
>>> 'Traceback' in trace
True
>>> "raise RuntimeError('yé')" in trace
True
```

pytoolbox.filesystem module

pytoolbox.flask module

pytoolbox.humanize module

pytoolbox.humanize.**naturalbitrate**(*bps*, *format='{sign}{value:.3g} {unit}'*, *scale=None*, *units=('bit/s', 'kb/s', 'Mb/s', 'Gb/s', 'Tb/s', 'Pb/s', 'Eb/s', 'Zb/s', 'Yb/s')*)

Return a human readable representation of a bit rate taking *bps* as the rate in bits/s.

The unit is taken from:

- The *scale* if not None (0=bit/s, 1=kb/s, 2=Mb/s, ...).
- The right scale from *units*.

Example usage

```
>>> naturalbitrate(-10)
'-10 bit/s'
>>> naturalbitrate(0.233)
'0.233 bit/s'
>>> naturalbitrate(69.5, format='{value:.2g} {unit}')
'70 bit/s'
>>> naturalbitrate(999.9, format='{value:.0f}{unit}')
'1000bit/s'
>>> naturalbitrate(1060)
'1.06 kb/s'
>>> naturalbitrate(3210837)
'3.21 Mb/s'
>>> naturalbitrate(16262710, units=['bps', 'Kbps'])
'1.63e+04 Kbps'
>>> naturalbitrate(3210837, scale=1, format='{value:.2f} {unit}')
'3210.84 kb/s'
```

pytoolbox.humanize.**naturalfilesize**(*bytes*, *system='nist'*, *format='{sign}{value:.3g} {unit}'*, *scale=None*, *args={'gnu': {'base': 1000, 'units': ('B', 'K', 'M', 'G', 'T', 'P', 'E', 'Z', 'Y')}, 'nist': {'base': 1024, 'units': ('B', 'kB', 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB', 'YB')}, 'si': {'base': 1000, 'units': ('B', 'KiB', 'MiB', 'GiB', 'TiB', 'PiB', 'EiB', 'ZiB', 'YiB')}}*)

Return a human readable representation of a *file* size taking *bytes* as the size in bytes.

The base and units taken from:

- The value in *args* with key *system* if not None.
- The *args* if *system* is None.

The unit is taken from:

- The *scale* if not None (0=Bytes, 1=KiB, 2=MiB, ...).
- The right scale from units previously retrieved from *args*.

Example usage

```
>>> naturalfilesize(-10)
'-10 B'
>>> naturalfilesize(0.233)
'0.233 B'
>>> naturalfilesize(1)
```

(continues on next page)

(continued from previous page)

```
'1 B'
>>> naturalfilesize(69.5, format='{value:.2g} {unit}')
'70 B'
>>> naturalfilesize(999.9, format='{value:.0f}{unit}')
'1000B'
>>> naturalfilesize(1060)
'1.04 kB'
>>> naturalfilesize(1060, system='si')
'1.06 KiB'
>>> naturalfilesize(3210837)
'3.06 MB'
>>> naturalfilesize(3210837, scale=1, format='{value:.2f} {unit}')
'3135.58 kB'
>>> naturalfilesize(16262710, system=None, args={'base': 1000, 'units': ['B', 'K
↪']})
'1.63e+04 K'
>>> naturalfilesize(314159265358979323846, system='gnu')
'314 E'
```

pytoolbox.humanize.**naturalfrequency**(*hertz*, *format='{sign}{value:.3g} {unit}'*, *scale=None*, *units=('Hz', 'kHz', 'MHz', 'GHz', 'THz', 'PHz', 'EHz', 'ZHz', 'YHz')*)

Return a human readable representation of a frequency taking *hertz* as the frequency in Hz.

The unit is taken from:

- The *scale* if not None (0=bit/s, 1=kb/s, 2=Mb/s, ...).
- The right scale from *units*.

Example usage

```
>>> naturalfrequency(-10)
'-10 Hz'
>>> naturalfrequency(0.233)
'0.233 Hz'
>>> naturalfrequency(69.5, format='{value:.2g} {unit}')
'70 Hz'
>>> naturalfrequency(999.9, format='{value:.0f}{unit}')
'1000Hz'
>>> naturalfrequency(1060)
'1.06 kHz'
>>> naturalfrequency(3210837)
'3.21 MHz'
>>> naturalfrequency(16262710, units=['Hertz', 'kilo Hertz'])
'1.63e+04 kilo Hertz'
>>> naturalfrequency(3210837, scale=1, format='{value:.2f} {unit}')
'3210.84 kHz'
```

pytoolbox.humanize.**naturalweight**(*grams*, *format='{sign}{value:.3g} {unit}'*, *scale=None*, *units=('g', 'Kg', 'T', 'KT', 'MT', 'GT')*)

Return a human readable representation of a weight in *grams*.

The unit is taken from: * The *scale* if not None (0=g, 1=Kg, 2=T, ...). * The right scale from *units*. **Example usage** >>> naturalweight(-10_000) '-10 Kg' >>> naturalweight(0.233) '0.233 g' >>> naturalweight(69.5, format='{value:.2g} {unit}') '70 g' >>> naturalweight(999.9, format='{value:.0f}{unit}') '1000g' >>> naturalweight(545_000) '545 Kg' >>> naturalweight(3_210_000_000) '3.21 KT' >>> naturalweight(1_620_000, units=['Grams', 'kilo Grams']) '1.62e+03 kilo Grams' >>> naturalweight(502456123, scale=2, format='{value:.2f} {unit}') '502.46 T'

pytoolbox.humanize.**natural_int_key**(*text*)

Function to be called as the key argument for list.sort() or sorted() in order to sort collections containing textual numbers on a more intuitive way.

Example usage

```
>>> sorted(['a26', 'a1', 'a4', 'a19', 'b2', 'a10', 'a3', 'b12'])
['a1', 'a10', 'a19', 'a26', 'a3', 'a4', 'b12', 'b2']
>>> sorted(['a26', 'a1', 'a4', 'a19', 'b2', 'a10', 'a3', 'b12'], key=natural_int_
↪key)
['a1', 'a3', 'a4', 'a10', 'a19', 'a26', 'b2', 'b12']
```

pytoolbox.itertools module

pytoolbox.juju module

pytoolbox.linux module

pytoolbox.linux.**get_kernel_config**(*release=None*, *fail=True*)

Return a JSON string with the GNU/Linux Kernel configuration.

Example usage

```
>>> config = get_kernel_config(fail=False)
>>> type(config)
<class 'dict'>
>>> not config or 'memory' in config
True
```

Error handling:

```
>>> get_kernel_config('0.0.1-generic', fail=False)
{}
```

pytoolbox.logging module

pytoolbox.module module

class pytoolbox.module.**All**(*globals_*)

Bases: object

__init__(*globals_*)

Initialize self. See help(type(self)) for accurate signature.

diff(*globals_*, *to_type=<class 'list'>*)

pytoolbox.pandas module

pytoolbox.private module

pytoolbox.regex module

pytoolbox.serialization module

pytoolbox.setuptools module

class pytoolbox.setuptools.**Disabled**(*dist*, ***kw*)
Bases: setuptools.Command

description = 'Do not run this.'

user_options = [('dummy=', 'd', 'dummy option to make setuptools happy')]

initialize_options()
Initialize options.

finalize_options()
Finalize options.

run()
A command's raison d'etre: carry out the action it exists to perform, controlled by the options initialized in 'initialize_options()', customized by other commands, the setup script, the command-line, and config files, and finalized in 'finalize_options()'. All terminal output and filesystem interaction should be done by 'run()'.

This method must be implemented by all command classes.

pytoolbox.signals module

pytoolbox.signals.**propagate_handler**(*signum*, *frame*)

pytoolbox.signals.**register_handler**(*signum*, *handler*, *append=True*, *reset=False*)

pytoolbox.signals.**register_callback**(*signum*, *callback*, *append=True*, *reset=False*, *args=None*, *kwargs=None*)

pytoolbox.states module

pytoolbox.string module

pytoolbox.string.**camel_to_dash**(*string*)
Convert camelCase to dashed-case.

pytoolbox.string.**camel_to_snake**(*string*)
Convert camelCase to snake_case.

pytoolbox.string.**dash_to_camel**(*string*)

pytoolbox.string.**snake_to_camel**(*string*)

pytoolbox.string.**filterjoin**(*items*, *sep=' '*, *keep=<function <lambda>>*)
Concatenate *items* with intervening occurrences of *sep*. Gracefully convert items to string and filter the items using the *keep* function.

pytoolbox.string.**to_lines**(*items*, *limit=80*, *start='\t'*, *template='{0} '*)
Convert items to string using *template*. Ensure lines length of maximum *limit*. Prefixing every line with *start*.

Example usage*

```
>>> some_culture = (  # Credits: https://en.wikipedia.org/wiki/Punched_card
...     "A legacy of the 80 column punched card format is that a display of 80␣
↪characters per"
...     " row was a common choice in the design of character-based terminals. As␣
↪of November"
```

(continues on next page)

(continued from previous page)

```
...     " 2011 some character interface defaults, such as the command prompt␣
↪window's width"
...     " in Microsoft Windows, remain set at 80 columns and some file formats,␣
↪such as FITS,"
...     " still use 80-character card images.")
```

Using default options:

```
>>> print(to_lines(some_culture.split(' ')))
    A legacy of the 80 column punched card format is that a display of 80
    characters per row was a common choice in the design of character-based
    terminals. As of November 2011 some character interface defaults, such as the
    command prompt window's width in Microsoft Windows, remain set at 80 columns
    and some file formats, such as FITS, still use 80-character card images.
```

Customizing output:

```
>>> print(to_lines(some_culture.split(' '), limit=42, start='> '))
> A legacy of the 80 column punched card
> format is that a display of 80
> characters per row was a common choice
> in the design of character-based
> terminals. As of November 2011 some
> character interface defaults, such as
> the command prompt window's width in
> Microsoft Windows, remain set at 80
> columns and some file formats, such as
> FITS, still use 80-character card
> images.
```

Displaying objects representation:

```
>>> class Item(object):
...     def __init__(self, value):
...         self.value = value
...
...     def __repr__(self):
...         return f'<Item value={self.value}>'
...
>>> print(to_lines((Item(i) for i in range(22)), limit=60, template='{0!r} '))
    <Item value=0> <Item value=1> <Item value=2>
    <Item value=3> <Item value=4> <Item value=5>
    <Item value=6> <Item value=7> <Item value=8>
    <Item value=9> <Item value=10> <Item value=11>
    <Item value=12> <Item value=13> <Item value=14>
    <Item value=15> <Item value=16> <Item value=17>
    <Item value=18> <Item value=19> <Item value=20>
    <Item value=21>
```

pytoolbox.subprocess module

pytoolbox.throttles module

Throttling classes implementing various throttling policies.

class pytoolbox.throttles.**TimeThrottle**(*min_time_delta*)

Bases: object

Time based throttling class.

```
>>> import datetime
>>> def slow_range(*args):
...     for i in range(*args):
...         time.sleep(0.5)
...         yield i
>>> t1, t2 = (TimeThrottle(t) for t in (datetime.timedelta(minutes=1), 0.2))
>>> list(t1.throttle_iterable((i, i) for i in range(10)))
[(0, 0), (9, 9)]
>>> list(t2.throttle_iterable(slow_range(3)))
[0, 1, 2]
```

__init__(*min_time_delta*)

Initialize self. See help(type(self)) for accurate signature.

is_throttled()

Return a boolean indicating if you should throttle.

throttle_iterable(*objects*, *callback=<function TimeThrottle.<lambda>>*)

Consume and skips some objects to yield them at defined *min_delay*. First and last objects are always returned.

- Set *callback* to a callable with the signature `is_throttled_args = callback(object)`. Used by subclasses.

class pytoolbox.throttles.**TimeAndRatioThrottle**(*min_ratio_delta*, *min_time_delta*, *max_time_delta*)

Bases: *pytoolbox.throttles.TimeThrottle*

Time and ratio based throttling class.

```
>>> import datetime
>>> def slow_range(*args):
...     for i in range(*args):
...         time.sleep(0.5)
...         yield i
>>> t1, t2 = (TimeAndRatioThrottle(0.3, t, 10*t) for t in (datetime.
↪timedelta(minutes=1), 0.4))
>>> list(t1.throttle_iterable(list(range(9)), lambda i: [i/9]))
[0, 8]
>>> list(t2.throttle_iterable(slow_range(9), lambda i: [i/9]))
[0, 3, 6, 8]
```

__init__(*min_ratio_delta*, *min_time_delta*, *max_time_delta*)

Initialize self. See help(type(self)) for accurate signature.

is_throttled(*ratio*)

Return a boolean indicating if you should throttle.

pytoolbox.types module

pytoolbox.types.**get_arguments_names**(*function*)

Return a list with arguments names.

```
>>> from pytoolbox import types
>>>
>>> get_arguments_names(get_arguments_names)
['function']
>>>
>>> def my_func(directory, a=1, *args, b, c=None, **kwargs):
...     pass
...
>>> get_arguments_names(my_func)
['directory', 'a', 'args', 'b', 'c', 'kwargs']
>>>
>>> get_arguments_names(types.get_subclasses)
['obj', 'nested']
>>> get_arguments_names(types.merge_bases_attribute)
['cls', 'attr_name', 'init', 'default', 'merge_func']
```

pytoolbox.types.**get_properties**(*obj*)

pytoolbox.types.**get_slots**(*obj*)

Return a set with the *__slots__* of the *obj* including all parent classes *__slots__*.

pytoolbox.types.**get_subclasses**(*obj*, *nested=True*)

Walk the inheritance tree of `obj`. Yield tuples with (class, subclasses).

Example usage

```
>>> class Root(object):
...     pass
...
>>> class NodeA(Root):
...     pass
...
>>> class NodeB(Root):
...     pass
...
>>> class NodeC(NodeA):
...     pass
...
>>> class NodeD(NodeA):
...     pass
...
>>> class NodeE(NodeD):
...     pass
...
>>> [(c.__name__, bool(s)) for c, s in get_subclasses(Root)]
[('NodeA', True), ('NodeC', False), ('NodeD', True), ('NodeE', False), ('NodeB',␣
↪False)]
>>> [(c.__name__, bool(s)) for c, s in get_subclasses(Root, nested=False)]
[('NodeA', True), ('NodeB', False)]
>>> [(c.__name__, bool(s)) for c, s in get_subclasses(NodeB)]
[]
>>> [(c.__name__, bool(s)) for c, s in get_subclasses(NodeD)]
[('NodeE', False)]
```

pytoolbox.types.**isiterable**(*obj*, *blacklist=(<class 'bytes'>, <class 'str'>)*)

Return `True` if the object is an iterable, but `False` for any class in *blacklist*.

Example usage

```
>>> from pytoolbox.unittest import asserts
>>> for obj in b'binary', 'unicode', 42:
...     asserts.false(isiterable(obj), obj)
>>> for obj in [], (), set(), iter({}.items()):
...     asserts.true(isiterable(obj), obj)
>>> isiterable({}, dict)
False
```

pytoolbox.types.**merge_annotations**(*cls: type*)

Merge annotations defined in all bases classes (using *__mro__*) into given *cls*.

Can be used as a decorator.

Example usage

```
>>> class Point2D(object):
...     x: int
...     y: int
...
>>> class Point3D(Point2D):
...     z: int
...
>>> class Point4D(Point3D, Point2D):
...     w: int
...
>>> @merge_annotations
... class Point4X(Point4D):
...     x: float
...     other: str
...
>>> assert Point2D.__annotations__ == {'x': int, 'y': int}
>>> assert Point3D.__annotations__ == {'z': int}
>>> assert Point4D.__annotations__ == {'w': int}
>>> assert Point4X.__annotations__ == {'x': float, 'y': int, 'z': int, 'w': int,
↪'other': str}
```

pytoolbox.types.**merge_bases_attribute**(*cls*, *attr_name*, *init*, *default*, *merge_func=<function <lambda>>*)

Merge all values of attribute defined in all bases classes (using *__mro__*). Return resulting value. Use default every time a class does not have given attribute.

Be careful, *merge_func* must be a pure function.

class pytoolbox.types.**DummyObject**(***kwargs*)

Bases: object

Easy way to generate a dynamic object with the attributes defined at instantiation.

Example usage

```
>>> obj = DummyObject(foo=42, bar=None)
>>> obj.foo
42
>>> obj.bar is None
True
```

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

class pytoolbox.types.**EchoObject**(*name*, ***attrs*)

Bases: object

Object that return any missing attribute as an instance of *EchoObject* with the name set to the Python expression used to access it. Also implements __getitem__. Some examples are worth hundred words...

Example usage

```
>>> from pytoolbox.unittest import asserts
>>> something = EchoObject('something', language='Python')
>>> something._name
'something'
>>> something.language
'Python'
>>> hasattr(something, 'everything')
True
>>> type(something.user.email)
<class 'pytoolbox.types.EchoObject'>
>>> str(something.user.first_name)
'something.user.first_name'
>>> str(something[0][None]['bar'])
"something[0][None]['bar']"
>>> str(something[0].node['foo'].x)
"something[0].node['foo'].x"
>>> str(something)
'something'
```

You can also define the class for the generated attributes:

```
>>> something.attr_class = list
>>> type(something.cool)
<class 'list'>
```

This class handles sub-classing appropriately:

```
>>> class MyEchoObject(EchoObject):
...     pass
>>>
>>> type(MyEchoObject('name').x.y.z)
<class 'pytoolbox.types.MyEchoObject'>
```

attr_class = None

__init__(*name*, ***attrs*)

Initialize self. See help(type(self)) for accurate signature.

class pytoolbox.types.**EchoDict**(*name*, ***items*)

Bases: dict

Dictionary that return any missing item as an instance of *EchoObject* with the name set to the Python expression used to access it. Some examples are worth hundred words...

Example usage

```
>>> context = EchoDict('context', language='Python')
>>> context._name
'context'
>>> context['language']
'Python'
>>> 'anything' in context
```

(continues on next page)

(continued from previous page)

```
True
>>> str(context['user'].first_name)
"context['user'].first_name"
>>> str(context[0][None]['bar'])
"context[0][None]['bar']"
>>> str(context[0].node['foo'].x)
"context[0].node['foo'].x"
```

You can also define the class for the generated items:

```
>>> context.item_class = set
>>> type(context['jet'])
<class 'set'>
```

item_class
alias of *EchoObject*

__init__(*name*, ***items*)
Initialize self. See help(type(self)) for accurate signature.

class pytoolbox.types.**MissingType**
Bases: object

pytoolbox.unittest module

pytoolbox.validation module

pytoolbox.virtualenv module

pytoolbox.voluptuous module

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

G

M

N

O

P

R

S

T

U

V

W

X

Z