puffotter

Release 0.9.4

Hermann Krumrey

Mar 31, 2020

CONTENTS

Contents:

CHAPTER

ONE

PUFFOTTER

1.1 puffotter package

1.1.1 Subpackages

puffotter.flask package

Subpackages

puffotter.flask.db package

Submodules

puffotter.flask.db.ApiKey module

class puffotter.flask.db.ApiKey.**ApiKey**(**args*, ***kwargs*)

Bases: *puffotter.flask.db.ModelMixin.ModelMixin*, sqlalchemy.ext.declarative.api.Model

Model that describes the 'api_keys' SQL table An ApiKey is used for API access using HTTP basic auth

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

Initializes the Model :param args: The constructor arguments :param kwargs: The constructor keyword arguments

creation_time

The time at which this API key was created as a UNIX timestamp

has_expired() → bool

Checks if the API key has expired. API Keys expire after 30 days :return: True if the key has expired, False otherwise

id

key_hash

The hash of the API key

user

The user associated with this API key

user_id

The ID of the user associated with this API key

verify_key(*key: str*) → bool
Checks if a given key is valid :param key: The key to check :return: True if the key is valid, False otherwise

puffotter.flask.db.ModelMixin module

class puffotter.flask.db.ModelMixin.**ModelMixin**
Bases: object

A mixin class that specifies a couple of methods all database models should implement

id = Column(None, Integer(), table=None, primary_key=True, nullable=False)
The ID is the primary key of the table and increments automatically

puffotter.flask.db.User module

class puffotter.flask.db.User.**User**(**args*, ***kwargs*)
Bases: *puffotter.flask.db.ModelMixin.ModelMixin*, sqlalchemy.ext.declarative.api.Model

Model that describes the 'users' SQL table A User stores a user's information, including their email address, username and password hash

__init__(**args*, ***kwargs*)
Initializes the Model :param args: The constructor arguments :param kwargs: The constructor keyword arguments

api_keys

confirmation_hash
The account's confirmation hash. This is the hash of a key emailed to the user. Only once the user follows the link in the email containing the key will their account be activated

confirmed
The account's confirmation status. Logins should be impossible as long as this value is False.

email
The user's email address

get_id() → str
Method required by flask-login :return: The user's ID as a unicode string

id

property is_active
Property required by flask-login :return: True

property is_anonymous
Property required by flask-login :return: True if the user is not confirmed, False otherwise

property is_authenticated
Property required by flask-login :return: True if the user is confirmed, False otherwise

password_hash
The user's hashed password, salted and hashed.

username
The user's username

verify_confirmation(*confirmation_key: str*) → bool
Verifies a confirmation key against the confirmation hash :param confirmation_key: The key to check :return: True if the key matches, False otherwise

verify_password(*password: str*) → bool
Verifies a password against the password hash :param password: The password to check :return: True if the password matches, False otherwise

Module contents

puffotter.flask.routes package

Subpackages

puffotter.flask.routes.api package

Submodules

puffotter.flask.routes.api.user_management module

puffotter.flask.routes.api.user_management.**define_blueprint**(*blueprint_name: str*) → flask.blueprints.Blueprint
Defines the blueprint for this route :param blueprint_name: The name of the blueprint :return: The blueprint

Module contents

Submodules

puffotter.flask.routes.decorators module

puffotter.flask.routes.decorators.**api**(*func: Callable*) → Callable
Decorator that handles common API patterns and ensures that the JSON response will always follow a certain pattern :param func: The function to wrap :return: The wrapper function

puffotter.flask.routes.decorators.**api_login_required**(*func: Callable*) → Callable
Decorator to make unauthorized API calls respond with JSON properly :param func: The function to wrap :return: The wrapped function

puffotter.flask.routes.static module

puffotter.flask.routes.static.**define_blueprint**(*blueprint_name: str*) → flask.blueprints.Blueprint
Defines the blueprint for this route :param blueprint_name: The name of the blueprint :return: The blueprint

puffotter.flask.routes.user_management module

puffotter.flask.routes.user_management.**define_blueprint**(*blueprint_name: str*) → flask.blueprints.Blueprint
Defines the blueprint for this route :param blueprint_name: The name of the blueprint :return: The blueprint

Module contents

puffotter.flask.routes.**blueprint_generators: List[Tuple[Callable[str, flask.blueprints.Blu**
Defines the functions used to create the various blueprints as well as their names

puffotter.flask.test package

Subpackages

puffotter.flask.test.db package

Submodules

puffotter.flask.test.db.TestApiKey module

class puffotter.flask.test.db.TestApiKey.**TestApiKey**(*methodName='runTest'*)
Bases: puffotter.flask.test.TestFramework._TestFramework

Class that tests the ApiKey database model

test_equality()
Tests checking equality for model objects :return: None

test_expiration()
Tests if the expiration of API keys works correctly :return: None

test_hashing()
Tests using the model objects as keys in a dictionary :return: None

test_json_representation()
Tests the JSON representation of the model :return: None

test_repr()
Tests the __repr__ method of the model class :return: None

test_string_representation()
Tests the string representation of the model :return: None

test_verifying_key()
Tests verifying an api key :return: None

puffotter.flask.test.db.TestModelMixin module

class puffotter.flask.test.db.TestModelMixin.**TestModelMixin**(*methodName='runTest'*)
Bases: puffotter.flask.test.TestFramework._TestFramework

Class that tests the ModelMixin class

test_enum_attributes()
Tests if the repr method handles enums correctly :return: None

puffotter.flask.test.db.TestUser module

class puffotter.flask.test.db.TestUser.**TestUser**(*methodName='runTest'*)
Bases: puffotter.flask.test.TestFramework._TestFramework

Class that tests the User database model

test_equality()
Tests checking equality for model objects :return: None

test_flask_properties()
Tests if the flask_login properties work as expected :return: None

test_hashing()
Tests using the model objects as keys in a dictionary :return: None

test_json_representation()
Tests the JSON representation of the model :return: None

test_repr()
Tests the __repr__ method of the model class :return: None

test_string_representation()
Tests the string representation of the model :return: None

test_verifying_password()
Tests verifying the password of a user :return: None

Module contents

puffotter.flask.test.misc package

Submodules

puffotter.flask.test.misc.TestApiCalls module

class puffotter.flask.test.misc.TestApiCalls.**TestApiCalls**(*methodName='runTest'*)
Bases: puffotter.flask.test.TestFramework._TestFramework

Class that tests various API calls

test_exception_in_api_route()
Tests if an exeption on an API route is correctly wrapped in a JSON response :return: None

test_expired_api_key()
Tests using an expired API key :return: None

test_non_base64_header()
Tests using a header that's not base64 encoded :return: None

test_random_exception()
Tests that the API routes catch any Exceptions without issue :return: None

test_unauthorized_call()
Tests and unauthorized API call :return: None

test_using_non_json_data()
Tests sending the data as something that's not JSON :return: None

puffotter.flask.test.misc.TestConfig module

class puffotter.flask.test.misc.TestConfig.**TestConfig**(*methodName='runTest'*)
Bases: puffotter.flask.test.TestFramework._TestFramework

Class that tests the config class

test_db_config()
Tests the database configuration :return: None

test_version()
Tests if the version is fetched correctly :return: None

puffotter.flask.test.misc.TestErrorHandling module

class puffotter.flask.test.misc.TestErrorHandling.**TestErrorHandling**(*methodName='runTest'*)
Bases: puffotter.flask.test.TestFramework._TestFramework

Class that tests the flask error handling

test_404()
Tests if a 404 error is handled correctly :return: None

test_exception()
Tests if unexpected exceptions are caught correctly :return: None

puffotter.flask.test.misc.TestInitialization module

class puffotter.flask.test.misc.TestInitialization.**TestInitialization**(*methodName='runTest'*)
Bases: puffotter.flask.test.TestFramework._TestFramework

Tests the initialization of the flask application

test_missing_environment_variables()
Tests if missing environment variables are detected correctly :return: None

test_missing_required_template()
Tests if missing template files are detected correctly :return: None

puffotter.flask.test.misc.TestWsgi module

class puffotter.flask.test.misc.TestWsgi.**TestWsgi**(*methodName='runTest'*)
Bases: puffotter.flask.test.TestFramework._TestFramework

Tests the WSGI server

test_starting_and_stopping_wsgi_server()
Tests starting and stopping the WSGI server :return: None

test_starting_background_tasks()
Tests starting background tasks :return: None

Module contents

puffotter.flask.test.routes package

Subpackages

puffotter.flask.test.routes.api package

Submodules

puffotter.flask.test.routes.api.TestApiKeyRoute module

class puffotter.flask.test.routes.api.TestApiKeyRoute.**TestApiKeyRoute**(*methodName='runTest'*)
Bases: puffotter.flask.test.TestFramework._TestFramework

Class that tests API-key related features

test_requesting_api_key()
Tests requesting an API key :return: None

test_requesting_invalid_api_keys()
Tests requesting API keys with invalid data :return: None

test_revoking_api_key()
Tests revoking an API key :return: None

test_unsuccessfully_revoking_api_key()
Tests unsuccessfully revoking an API key :return: None

Module contents

Submodules

puffotter.flask.test.routes.TestForgotRoute module

class puffotter.flask.test.routes.TestForgotRoute.**TestForgotRoute**(*methodName='runTest'*)
Bases: puffotter.flask.test.TestFramework._TestFramework

Class that tests password reset features

test_invalid_recaptcha()
Tests that invalid ReCaptcha responses are handled correctly :return: None

test_page_get()
Tests getting the page :return: None

test_resetting_password()
Tests successfully resetting a password :return: None

test_unsuccessfully_resetting_password()
Tests unsuccessfully resetting a password :return: None

puffotter.flask.test.routes.TestLoginRoute module

class puffotter.flask.test.routes.TestLoginRoute.**TestLoginRoute**(*methodName='runTest'*)
Bases: puffotter.flask.test.TestFramework._TestFramework

Class that tests log-in features

test_invalid_login_attempts()
Tests trying to log in with invalid credentials etc :return: None

test_logging_in_and_out()
Tests logging in successfully, then once more, then logging out :return: None

test_page_get()
Tests getting the page :return: None

puffotter.flask.test.routes.TestProfileRoute module

class puffotter.flask.test.routes.TestProfileRoute.**TestProfileRoute**(*methodName='runTest'*)
Bases: puffotter.flask.test.TestFramework._TestFramework

Class that tests profile features

test_changing_password()
Tests changing a password :return: None

test_page_get()
Tests getting the page :return: None

test_unsuccessful_password_change()
Tests unsuccessfully changing a password :return: None

test_unsuccessful_user_delete()
Tests unsuccessfully deleting a user :return: None

test_user_delete()
Tests deleting a user :return: None

puffotter.flask.test.routes.TestRegisterRoute module

class puffotter.flask.test.routes.TestRegisterRoute.**TestRegisterRoute**(*methodName='runTest'*)
Bases: puffotter.flask.test.TestFramework._TestFramework

Class that tests registration features

test_confirming()
Tests confirming a user :return: None

test_invalid_confirm()
Tests invalid confirmations :return: None

test_invalid_recaptcha()
Tests that invalid ReCaptcha responses are handled correctly :return: None

test_invalid_registrations()
Tests registering using invalid parameters :return: None

test_page_get()
Tests getting the page :return: None

test_registering_user()
Tests registering a new user :return: None

puffotter.flask.test.routes.TestStaticRoutes module

class puffotter.flask.test.routes.TestStaticRoutes.**TestStaticRoutes**(*methodName='runTest'*)
Bases: puffotter.flask.test.TestFramework._TestFramework

Class that tests static pages

test_get_about()
Tests getting the about page :return: None

test_get_index()
Tests getting the index page :return: None

test_get_privacy()
Tests getting the privacy page :return: None

Module contents

Submodules

puffotter.flask.test.TestFramework module

Module contents

Submodules

puffotter.flask.Config module

class puffotter.flask.Config.**Config**
Bases: object

Class that keeps track of configuration data The class attributes should only be called after running load_config

API_VERSION: str = '1'
The API Version

DB_MODE: str = None
The database mode (for example 'sqlite' or 'mysql')

DB_URI: str = None
The database URI to use for database operations

FLASK_PORT: int = 8000
The port to use when serving the flask application

FLASK_SECRET: str = None
The flask secret key

LOGGING_PATH: str = None
The path to the logging file

MAX_API_KEY_AGE: int = 2592000
The maximum age for API keys

MAX_USERNAME_LENGTH: int = 12
The maximum length of usernames

MIN_USERNAME_LENGTH: int = 1
The Minimum length for usernames

RECAPTCHA_SECRET_KEY: str = None
Google ReCaptcha secret key for bot detection

RECAPTCHA_SITE_KEY: str = None
Google ReCaptcha site key for bot detection

REQUIRED_TEMPLATES: Dict[str, str] = {'about': 'static/about.html', 'error_page': 's
Specifies required template files

SENTRY_DSN: str = None
The sentry DSN used for error logging

SMTP_ADDRESS: str = None
The SMPT Address used for sending emails

SMTP_HOST: str = None
The SMPT Host used for sending emails

SMTP_PASSWORD: str = None
The SMPT Password used for sending emails

SMTP_PORT: int = None
The SMPT Port used for sending emails

STRINGS: Dict[str, str] = {'401_message': 'You are not logged in', '500_message': 'T
Dictionary that defines various strings used in the application. Makes it easier to use custom phrases.

TEMPLATE_EXTRAS: Dict[str, Callable[Dict[str, Any]]] = {'about': <function Config.<la
This can be used to provide the template rendering engine additional parameters, which may be necessary when adding UI elements. This is done with functions that don't expect any input and return a dictionary of keys and values to be passed to the template rendering engine

TESTING: bool = None
Whether or not testing is enabled

VERSION: str = None
The current version of the application

classmethod load_config(*root_path: str*, *module_name: str*, *sentry_dsn: str*)
Loads the configuration from environment variables :param root_path: The root path of the application :param module_name: The name of the project's module :param sentry_dsn: The sentry DSN used for error logging :return: None

puffotter.flask.base module

puffotter.flask.base.**app = <Flask 'puffotter.flask.base'>**
The Flask App

puffotter.flask.base.**db = <SQLAlchemy engine=None>**
The SQLAlchemy database connection

puffotter.flask.base.**login_manager = <flask_login.login_manager.LoginManager object>**
The Flask-Login Login Manager

puffotter.flask.enums module

class puffotter.flask.enums.**AlertSeverity**
Bases: enum.Enum

Enumeration that defines the various levels of severity an alert can have

DANGER = 'danger'
Translates to a red alert

INFO = 'info'
Translates to a blue alert

SUCCESS = 'success'
Translates to a green alert

WARNING = 'warning'
Translates to a yellow alert

puffotter.flask.exceptions module

exception puffotter.flask.exceptions.**ApiException**(*reason: str*, *status_code: int*)
Bases: Exception

Api raised when an API-related exception occurs

__init__(*reason: str*, *status_code: int*)
Initializes the exception :param reason: The reason the API Exception was raised :param status_code: The status code associated with the exception

puffotter.flask.initialize module

puffotter.flask.initialize.**CREATED_BLUEPRINTS = []**
Keeps track of created blueprint names. This is necessary for unit testing with nose, because duplicate blueprint names will cause errors.

puffotter.flask.initialize.**init_flask**(*module_name: str, sentry_dsn: str, root_path: str, config: Type[puffotter.flask.Config.Config], models: List[Type[sqlalchemy.ext.declarative.api.Model]], blueprint_generators: List[Tuple[Callable[str, flask.blueprints.Blueprint], str]]*)
Initializes the flask application :param module_name: The name of the module :param sentry_dsn: The sentry DSN used for error logging :param root_path: The root path of the flask application :param config: The Config class to use for configuration :param models: The database models to create :param blueprint_generators: Tuples that contain a function that generates

a blueprint and the name of the blueprint

Returns None

puffotter.flask.wsgi module

puffotter.flask.wsgi.**start_server**(*config: Type[puffotter.flask.Config.Config], task_definitions: Dict[str, Tuple[int, Callable]]*)
Starts the flask application using a cheroot WSGI server :param config: The configuration to use :param task_definitions: The background tasks, consisting of:

- the name of the task
- the time to wait before running the task again
- a function that takes no arguments that executes the task

Returns None

Module contents

puffotter.test package

Submodules

puffotter.test.TestCrypto module

class puffotter.test.TestCrypto.**TestCrypto**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests cryptographical functions

test_hashing()
Tests that passwords can be hashed successfully :return: None

test_hashing_strings()
Tests that password hashing and verifying works with string as well :return: None

test_salt()
Tests that password hashes are salted :return: None

test_verifying_with_invalid_hash()
Tests that attempting to verify a password with an incorrectly formatted hash will return False :return: None

puffotter.test.TestOs module

class puffotter.test.TestOs.**TestCrypto**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests os functions

test_listdir()
Tests the listdir function :return: None

puffotter.test.TestUnits module

class puffotter.test.TestUnits.**TestCrypto**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests cryptographical functions

test_converting_bytes_to_human_readable()
Tests that passwords can be hashed successfully :return: None

Module contents

1.1.2 Submodules

1.1.3 puffotter.crypto module

puffotter.crypto.**generate_hash**(*password: str*) → str
Salts and hashes a password to generate a hash for storage in a database :param password: The password to hash :return: The hash of the password

puffotter.crypto.**generate_random**(*length: int*) → str
Generates a random byte string consisting of alphanumeric characters Thanks @ Albert (https://stackoverflow.com/users/281021/albert) https://stackoverflow.com/questions/2257441 :param length: The length of the string to generate :return: The generated random byte string

puffotter.crypto.**verify_password**(*password: str*, *hashed: str*)
Verifies that a password matches a given hash :param password: The password to verify :param hashed: The hash to verify the password against :return: True if the password matches, otherwise False

1.1.4 puffotter.env module

puffotter.env.**load_env_file**(*path: str = '.env'*)
Loads an environment file into os.environ :param path: THe path to the environment file :return: None

1.1.5 puffotter.graphql module

class puffotter.graphql.**GraphQlClient**(*api_url: str*)
Bases: object

A simple API wrapper for GraphQL APIs

__init__(*api_url: str*)
Initializes the GraphQL API wrapper :param api_url: The API endpoint URL

query(*query_string: str, variables: Optional[Dict[str, Any]]*) → Optional[Dict[str, Any]]
Executes a GraphQL query :param query_string: The query string to use :param variables: The variables to send :return: The response JSON, or None if an error occurred.

1.1.6 puffotter.imap module

puffotter.imap.**get_inbox_count**(*imap_server: str, imap_address: str, imap_password: str, imap_port: int = 993*) → int
Checks the amount of emails in an IMAP inbox :param imap_server: The IMAP server to use :param imap_address: The IMAP address to use :param imap_password: The IMAP password to use :param imap_port: The IMAP port to use :return: The amount of emails

1.1.7 puffotter.init module

puffotter.init.**argparse_add_logfile**(*parser: argparse.ArgumentParser*)
Adds the –logfile argument to the argument parser :param parser: The argument parser to modify :return: None

puffotter.init.**argparse_add_verbosity**(*parser: argparse.ArgumentParser*)
Adds –quiet, –verbose and –debug parameters to an ArgumentParser :param parser: the parser to which to add those flags :return: None

puffotter.init.**cli_start**(*main_func: Union[Callable[None], Callable[argparse.Namespace, None], Callable[[argparse.Namespace, logging.Logger], None]], arg_parser: argparse.ArgumentParser, exit_msg: str = 'Goodbye', package_name: Optional[str] = None, sentry_dsn: Optional[str] = None, release_name: Optional[str] = None*)
Starts a program and sets up logging, as well as sentry error tracking :param main_func: The main function to call :param arg_parser: The argument parser to use :param exit_msg: The message printed when the program's execution is

stopped using a keyboard interrupt

Parameters

- **package_name** – The package name of the application
- **sentry_dsn** – The sentry DSN to use
- **release_name** – The name of the release

Returns None

puffotter.init.**setup_logging**(*args: argparse.Namespace, package_name: Optional[str]*)
Sets up logging for the provided arguments :param args: The CLI arguments :param package_name: The package name :return: None

1.1.8 puffotter.json module

puffotter.json.**jsonify_models**(*data: Dict[str, Any], deep: bool = True*) → Dict[str, Any]
Serializes a dictionary and calls the __json__() method on all objects that have one. :param data: The data to serialize :param deep: Indicates whether or not child models are included.

If False, only IDs will be included.

Returns The serialized data

1.1.9 puffotter.logging module

class puffotter.logging.**ColorLogger**(*logger: logging.Logger, debug_bg: str = 'x1b[47m', debug_fg: str = 'x1b[30m', info_bg: str = 'x1b[40m', info_fg: str = 'x1b[37m', warning_bg: str = 'x1b[43m', warning_fg: str = 'x1b[30m', error_bg: str = 'x1b[101m', error_fg: str = 'x1b[30m'*)
Bases: object

__init__(*logger: logging.Logger, debug_bg: str = '\x1b[47m', debug_fg: str = '\x1b[30m', info_bg: str = '\x1b[40m', info_fg: str = '\x1b[37m', warning_bg: str = '\x1b[43m', warning_fg: str = '\x1b[30m', error_bg: str = '\x1b[101m', error_fg: str = '\x1b[30m'*)
Initialize self. See help(type(self)) for accurate signature.

debug(*message: str, bg: Optional[str] = None, fg: Optional[str] = None*)
Logs a message at DEBUG level :param message: The message to log :param bg: Overrides the default background style :param fg: Overrides the default foreground style :return: None

error(*message: str, bg: Optional[str] = None, fg: Optional[str] = None*)
Logs a message at ERROR level :param message: The message to log :param bg: Overrides the default background style :param fg: Overrides the default foreground style :return: None

info(*message: str, bg: Optional[str] = None, fg: Optional[str] = None*)
Logs a message at INFO level :param message: The message to log :param bg: Overrides the default background style :param fg: Overrides the default foreground style :return: None

warning(*message: str, bg: Optional[str] = None, fg: Optional[str] = None*)
Logs a message at WARNING level :param message: The message to log :param bg: Overrides the default background style :param fg: Overrides the default foreground style :return: None

1.1.10 puffotter.os module

puffotter.os.**create_file**(*path: str*)
Creates an empty file :param path: The path to the file :return: None

puffotter.os.**get_ext**(*filename: str*) → Optional[str]
Gets the file extension of a file :param filename: The filename for which to get the file extension :return: The file extension or None if the file has no extension

puffotter.os.**listdir**(*path: str, no_files: bool = False, no_dirs: bool = False, no_dot: bool = True*) → List[Tuple[str, str]]
Improves on the standard os.listdir function. By default, files and directories starting with . are ignored and one can disable checking for files or directories. Instead of just the filenames, the function returns tuples of filenames and relative file paths. :param path: The path of which to list the contents :param no_files: If set to True, will ignore files :param no_dirs: If set to True, will ignore directories :param no_dot: If set to True, will ignore files starting with . :return: A sorted list of the contents of the directory, consisting of

tuples of the file/directory name and their relative path.

puffotter.os.**makedirs**(*path: str*, *delete_before: bool = False*)
A more pleasant to use makedirs function. Only calls os.makedirs if the directory does not already exist :param path: The path to the directory to create :param delete_before: If True, deletes the directory beforehand,

if it exists

Returns None

puffotter.os.**replace_illegal_ntfs_chars**(*string: str*) → str
Replaces illegal characters in NTFS file systems with appropriate substitutes :param string: Te string in which to replace the illegal characters :return: The sanitized string

1.1.11 puffotter.print module

puffotter.print.**pprint**(**objects: str*, *sep: str = ' '*, *end: str = '\n'*, *file: IO = <_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *flush: bool = False*, *fg: Optional[str] = None*, *bg: Optional[str] = None*)
Function that extends the print function :return: None

1.1.12 puffotter.prompt module

puffotter.prompt.**prompt**(*prompt_text: str = ''*, *default: Optional[Any] = None*, *_type: Callable[[str], Any] = <class 'str'>*, *required: bool = True*, *choices: Optional[Set[str]] = None*) → Optional[Any]
Generic prompt with configuration options :param prompt_text: The text to display before the prompt :param default: A default value to use if the user responds with '' :param _type: The type of the object prompted. Must take a single string

as a parameter

Parameters

- **required** – Whether or not a response is required
- **choices** – Valid choices for the prompt

Returns The prompt result. May be None if required is False

puffotter.prompt.**prompt_comma_list**(*message: str*, *primitive_type: Callable[[str], Any] = <class 'str'>*, *min_count: int = 0*, *no_empty: bool = True*, *default: Optional[List[Any]] = None*) → List[Any]
Prompts the user for a comma-separated list :param message: The message to display :param primitive_type: The primitive type of the elements in the list :param min_count: The minimum amount of elements to be provided by the user :param no_empty: Removes any empty strings :param default: A default value :return: The result of the prompt

puffotter.prompt.**selection_prompt**(*objects: List[object]*) → List[object]
Prompts the user for a selection from a list of objects :param objects: The objects to show :return: The selection of objects

puffotter.prompt.**yn_prompt**(*message: str*, *make_sure: bool = True*, *case_sensitive: bool = False*) → bool
Creates a yes/no prompt :param message: The message to display :param make_sure: Continuously prompts if the response is neither

'y' or 'n' until it is. If false, every input besides 'y' will result in the return value being False

Parameters **case_sensitive** – Whether or not the prompt should be case-sensitive

Returns True if the user specified 'y', else False

1.1.13 puffotter.recaptcha module

puffotter.recaptcha.**verify_recaptcha**(*client_ip: str*, *recaptcha_response: str*, *secret_key: str*) → bool

Verifies a recaptcha response. If the recaptcha response originates from a local address, this method will always return True. :param client_ip: The IP Address of the client solving the captcha :param recaptcha_response: the recaptcha response to verify :param secret_key: the recaptcha secret key :return: True if the recaptcha response was correct, False otherwise

1.1.14 puffotter.requests module

puffotter.requests.**aggressive_request**(*url: str*) → str

Handles GET requests while analyzing status codes :param url: The URL to get :return: The response text

1.1.15 puffotter.smtp module

puffotter.smtp.**send_email**(*address: str*, *title: str*, *message: str*, *smtp_server: str*, *smtp_address: str*, *smtp_password: str*, *smtp_port: int = 587*)

Sends an HTML email message using SMTP :param address: The address to send to :param title: The email's title :param message: The message to send :param smtp_server: The SMTP server to use :param smtp_address: The SMTP address to use :param smtp_password: The SMTP password to use :param smtp_port: The SMTP port to use :return: None

1.1.16 puffotter.subprocess module

puffotter.subprocess.**execute_command**(*command: List[str]*) → int

Executes a command :param command: The command to execute :return: The status code

1.1.17 puffotter.testutils module

1.1.18 puffotter.units module

puffotter.units.**byte_string_to_byte_count**(*byte_string: str*) → int

Converts a string representing bytes to a number of bytes. For example: "500K" -> 500 000

"2.5M" -> 2 500 000 "10GB" -> 10 000 000 000 "30kb/s" -> 30 000

Parameters **byte_string** – The string to convert

Returns The amount of bytes

puffotter.units.**human_readable_bytes**(*bytecount: int*, *base_1024: bool = False*, *remove_trailing_zeroes: bool = True*) → str

Converts an amount of bytes into a human-readable string :param bytecount: The bytes to convert :param base_1024: Whether or not to use 1024 as base (for mebibytes etc) :param remove_trailing_zeroes: If set to True, will remove any trailing

zeroes from the string

Returns The human-readable string

1.1.19 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

F

G

H

Q

R

S

T

U

V

W

Y