fishbase Documentation

Release 1.1.13

David Yi

Jun 11, 2019

Contents

fishbase 是由我们自主开发和整理的一套 Python 基础函数库。

自 2016/3 初次发布以来，我们坚持不断更新，先后发布了 20 余个版本。近一年来，我们逐步形成每月更新 1 到 2 个版本的频率，抽象出了很多通用的方法，主要分为以下模块：

模块	功能函数
fish_common	基本函数包
fish_crypt	加密数据函数包
fish_csv	csv 处理增强函数包
fish_data	数据信息处理函数包，含银行卡、身份证等
fish_date	日期处理增强函数包
fish_file	文件处理增强函数包
fish_logger	日志记录增强函数包
fish_project	project 目录结构生成函数包
fish_random	随机数据生成函数包
fish_system	系统增强函数包

CHAPTER 1

安装

```
# 通过 pip 进行安装或者更新
pip install -U fishbase
```

CHAPTER 2

fishbase 能干什么?

2.1 获取当前系统类型

```
>>> from fishbase.fish_system import get_platform
>>> print('current os:', get_platform())
current os: osx
```

2.2 获取文件的绝对路径

```
>>> from fishbase.fish_common import find_files
>>> print(get_abs_filename_with_sub_path('/etc', 'hosts'))
(True, '/etc/hosts')
```

2.3 根据时间戳获取时间间隔

```
>>> from fishbase.fish_date import get_time_interval
>>> print(get_time_interval(1548575829,1548476921))
{'days': 1, 'hours': 3, 'minutes': 28, 'seconds': 28}
```

2.4 生成随机数据

```
>>> from fishbase.fish_random import gen_random_id_card
>>> # 随机生成一个身份证号
>>> print(gen_random_id_card())
['3101091986******47']
```

(continues on next page)

(continued from previous page)

```
>>> from fishbase.fish_random import gen_random_bank_card
>>> # 随机生成一个中国银行的信用卡卡号
>>> print(gen_random_bank_card('中国银行', 'CC'))
625907379******1
```

2.5 创建项目结构

```
>>> import os
>>> from fishbase.fish_project import init_project_by_yml
>>> package_yml = '''
... project: hellopackage
... tree:
...     - README.md
...     - requirements.txt
...     - setup.py
... '''
>>> # 通过 yml 文件创建一个项目结构
>>> init_project_by_yml(package_yml, '.')
>>> print(os.listdir('./hellopackage'))
['requirements.txt', 'README.md', 'setup.py']
```

2.5.1 更新记录

2019.6.11 v1.1.13

- #235, common, add function `RMBConversion.an2cn()`, `RMBConversion.cn2an()` doc and unittest;

2019.5.28 v1.1.12

- #232, data, edit function `IdCard.get_zone_info()`, `IdCard.get_areanote_info()`, `IdCard.get_province_info()`, `CardBin.get_bank_info()`, `CardBin.get_cardbin_info()`, optimize;

2019.5.14 v1.1.11

- #229, random, edit function *`fish_random.gen_random_bank_card()`* optimize;

2019.4.30 v1.1.10

- #226, file, add function *`fish_file.get_file_encoding()`*, doc and unittest;

2019.4.15 v1.1.9

- #222, common, edit function `fish_logger.conf_as_dict()`, optimize
- #221, logger, edit function *`fish_logger.set_log_file()`*, optimize

2019.4.1 v1.1.8

- #218, file, edit function *fish_file.get_abs_filename_with_sub_path()*, *fish_file.check_sub_path_create()*, optimize
- #215, common, add function :meth:‘fish_common.DeserializeInstance, doc and unittest;

2019.3.19 v1.1.7

- #212, common, edit function *fish_common.conf_as_dict()*, *fish_common.find_files()*, *fish_common.yaml_conf_as_dict()*, optimize
- #215, common, edit function *fish_common.serialize_instance()*, optimize doc and add unittest

2019.1.22 v1.1.6

- #192, data, add function `fish_data.IdCard.get_cn_idcard()`, `fish_data.IdCard.get_note_by_province()`, doc and unittest;
- #190, random, edit function `fish_random.gen_float_by_range()`, optimize;
- #152, random, edit function `fish_common.GetMD5()` `fish_common.GetSha256()` `fish_common.splice_url_params()` *fish_common.sorted_list_from_dict()* `fish_common.is_contain_special_char()` `fish_common.if_any_elements_is_space()` `fish_common.remove_duplicate_elements()` `fish_common.sorted_objs_by_attr()` `fish_common.get_group_list_data()` `fish_common.if_any_elements_is_letter()` `fish_common.transform_hump_to_underline()`, optimize;
- #204, random, edit function *fish_random.gen_random_id_card()*, *fish_random.gen_random_address()*, *fish_random.gen_random_bank_card()*, *fish_random.gen_random_company_name()*, *fish_random.gen_random_float()*, *fish_random.gen_random_mobile()*, *fish_random.gen_random_name()*,optimize;
- #200, random, edit function *fish_random.gen_random_str()*, optimize;
- #200, crypt, move fish_common.FishMD5 to *fish_crypt.FishMD5()*, move fish_common.Base64 to `fish_crypt.Base64()`, move fish_common.FishSha256 to *fish_crypt.FishSha256()*

2018.12.31 v1.1.5

- #171, random, add function `fish_random.gen_company_name()`, doc and unittest;
- #165, random, add function `fish_random.gen_id()`, doc and unittest;
- #172, random, add function `fish_random.gen_bank_card()`, doc and unittest;
- #170, random, add function `fish_random.gen_address()`, doc and unittest;
- #173, random, add function `fish_random.get_random_zone_name()`, doc and unittest;
- #162, random, add function `fish_random.gen_float_by_range()`, doc and unittest;
- #166, random, add function `fish_random.gen_mobile()`, doc and unittest;
- #171, random, add function `fish_random.gen_name()`, doc and unittest;
- #163, random, add function `fish_random.gen_string_by_range()`, doc and unittest;
- #164, common, add function *fish_date.GetRandomTime.gen_date_by_range()*, doc and unittest;

- #142, common, edit function *fish_date.GetRandomTime.gen_date_by_year()*, doc and unittest;

2018.12.14 v1.1.4

- #142, common, add function `fish_date.GetRandomTime.random_date_str()`, doc and unittest;
- #126, csv, add function *fish_csv.dict2csv()*, *fish_csv.csv2dict()*, *fish_csv.list2csv()*, doc and unittest;

2018.12.10 v1.1.3

- #137, data, add function `fish_data.is_valid_id_number()`, doc and unittest;
- #98, common, add function *fish_common.yaml_conf_as_dict()*, doc and unittest;
- #100, common, add class `fish_common.GetSha256()`, doc and unittest;
- #116, date, add class *fish_date.FishDateTimeFormat()*, doc and unittest;
- #80, common, add function *fish_common.find_same_between_dicts()*, doc and unittest;

2018.10.27 v1.1.2

- #99, common, add function `fish_common.GetMD5.hmac_md5()`, doc and unittest;

2018.9.23 v1.1.1

- #115, common, add function *fish_common.get_random_str()*, optimize;
- #114, common, add function `fish_common.transform_hump_to_underline()`, doc and unittest;
- #101, date, add function *fish_date.transform_datetime_to_unix()*, doc and unittest;

2018.9.3 v1.1.0

- #74, common, add function `fish_common.get_group_list_data()`, doc and unittest;
- #89, common, add function *fish_common.get_sub_dict()*, doc and unittest;
- #90, common, add function *fish_date.get_time_interval()*, doc and unittest;
- #93, common, add function *fish_date.transform_unix_to_datetime()*, doc and unittest;
- #82, project, add function *fish_project.init_project_by_yml()*, doc and unittest;

2018.8.2 v1.0.16

- #87, date, add function *fish_date.GetRandomTime()*, doc and unittest;
- #94, csv, edit function *fish_csv.csv_file_to_list()*, doc and unittest;
- #94, common, edit function *fish_common.conf_as_dict()*, doc and unittest;

2018.7.11 v1.0.15

- #36, common, edit function `fish_common.is_contain_special_char()`, change function name;
- #62, common, edit function `fish_common.if_any_elements_is_space()`, optimize, doc and unittest;
- #78, optimize change_log;
- #67, common, edit function `fish_common.splice_url_params()`, optimize;
- #63 and #77, common, add function `fish_common.remove_duplicate_elements()`, doc and unittest;
- #64 common, add function `fish_common.sorted_objs_by_attr()`, doc and unittest;
- #79 common, add function *`fish_common.get_query_param_from_url()`*, doc and unittest;
- #83 common, edit function *`fish_common.conf_as_dict()`*, optimize;

2018.6.27 v1.0.14

- 19046, setup, edit setup.py to add long description etc., the package detail;
- issue ID use directly on github
- #36, common, add function `fish_common.check_str()`, doc and unittest;
- #38, common, add function *`fish_common.find_files()`*, doc and unittest;
- #37, date, add function *`fish_date.get_years()`*, doc and unittest;
- #27, common, add function `fish_common.hmac_sha256()`, doc and unittest;
- #61, date, edit function *`fish_date.get_date_range()`*, optimize, doc and unittest;
- #57, common, edit function `fish_common.GetMD5.string()`, optimize;
- #59, common, add function `fish_common.Base64()`, doc and unittest;
- #51, common, add function *`fish_common.get_random_str()`*, doc and unittest;

2018.6.6 v1.0.13

- 19037, common and system, function `check_platform()` move to fish_system 中，rename to *`fish_system.get_platform()`*;
- 19038, common, add function *`fish_common.get_uuid()`*, edit `fish_common.get_time_uuid`, add doc and unittest;
- 19039, logger, edit function *`fish_logger.set_log_file()`* by `class SafeFileHandler()`, prevent the multi process delete log file error;
- 19040, file, edit function *`fish_file.get_abs_filename_with_sub_path()`*, thanks to Wu Yanan;
- 19041, file, delete function `check_kind_path_file()`;
- 19042, file, edit function *`fish_file.check_sub_path_create()`*, optimize, doc and unittest;
- 19043, common, edit function *`fish_common.sorted_list_from_dict()`*, optimize, doc and unittest;
- 19044, file, remove `auto_add_file_ext()`;
- 19045, file, remove `get_abs_filename_with_sub_path_module()`;

2018.5.21 v1.0.12

- 19035, rename package 'fish_base' to 'fishbase'

2018.5.18 v1.0.11

- 19011, 从19011开始编号，ok
- 19015, `common conf_as_dict()` 增加 docstring 说明， ok
- 19016, 开始测试使用 sphinx 来组织 api 说明文档，ok
- 19017, 将 `conf_as_dict()` 说明加入到 doc 中，ok
- 19018, __init__.py 中的 `get_ver()` 返回版本号功能简化，ok
- 19019, common `class SingleTon()` 增加 docstring 说明，ok
- 19020, csv `csv_file_to_list()` 增加 docstring 说明，ok
- 19021, common 重新声明为 fish_common, csv 重新生命为 fish_csv, 所有包带 fish 前缀，ok
- 19022, sphinx doc 的 theme 修改为 rtd theme，https://sphinx-rtd-theme.readthedocs.io/en/latest/ , ok
- 19023, logger `set_log_file()` 增加 docstring 说明，ok
- 19024, fish_file 函数加入 docstring 说明，ok
- 19025, common, 去除 `get_md5()` 函数，ok
- 19026, common, 增加 `class GetMD5`，增加字符串、小文件、大文件三种类型的 md5计算，ok
- 19027, test, 修改原来的 unittest 部分，完善对于 common 函数的单元测试，ok
- 19028, common, `conf_as_dict()` 逻辑修改，更加严密，ok
- 19029, common, 增加 `json_contained()` 函数，判断两个 json 是否有包含关系，ok
- 19030, common, 增加 `splice_url_params()` 函数；ok
- 19031, 项目，增加 requirements.txt; ok
- 19032, 项目，增加 .travis.yml, 支持持续集成测试; ok
- 19033, 项目，增加对于 coveralls.io 的支持，监视 ut 的覆盖率; 本地 python 2.7.15 测试通过; ok
- 19034, 项目，修改 __init__.py 和 setup.py 中对于 __version__ 的用法; ok

2018.3.20 v1.0.10

- 19006, 增加，`get_time_uuid()`, 获得带时间戳的流水号；ok
- 19007, 增加，`if_any_elements_is_space()`, 判断参数列表是否存在 None 或空字符串或空格字符串；ok
- 19008, common，增加 `conf_as_dict()`,读入配置文件，返回根据配置文件内容生成的字典类型变量; ok
- 11001, 整体结构和开发方法调整;
- 11002, 增加 csv 功能模块，增加函数 `csv_file_to_list()`; ok
- 11003, fish_file 模块修改为 file，目前向下兼容保留 fish_file; ok
- 11004, file 模块的 `get_abs_filename_with_sub_path()` 修改；ok

- 11005, fish_date 模块修改为 date, demo/demo_date.py 演示用法；ok
- 11006, 安装包的安装程序 setup.py 中 setup.py 引入源的修改；ok
- 11007, pip 安装时候支持自动安装 `python-dateutil` 包; ok
- 11008, `check_platform()` 归入到 system 包
- 11009, csv, `csv_file_to_list()` 函数增加过滤空行功能；ok
- 11010, logger, log 相关代码优化简化; ok
- 11011, demo, 将原来 test 下的 test log 程序移动到 demo 路径下; ok
- 11013, demo, `common.conf_as_dict()` 的 demo 例子完善；ok
- 11014, common, `conf_as_dict()` 增加返回内容，字典长度；ok
- 11015, common 增加 `class SingleTon`，单例的基础类；ok

2.6 API 函数列表

可以到以下单元中查找具体的函数列表和使用说明。

2.6.1 `fish_common` 基本函数包

fish_common.conf_as_dict(conf_filename[, ...])	读入 ini 配置文件，返回根据配置文件内容生成的字典类型变量;
fish_common.SingleTon	申明一个单例类，可以作为需要单例类时候申明用的父类
fish_common.get_uuid(kind)	获得不重复的 uuid，可以是包含时间戳的 uuid，也可以是完全随机的；基于 Python 的 uuid 类进行封装和扩展;
fish_common.has_space_element(source)	判断对象中的元素，如果存在 None 或空字符串，则返回 True, 否则返回 False, 支持字典、列表和元组
fish_common.if_json_contain(left_json, ...)	判断一个 json 是否包含另外一个 json 的 key，并且 value 相等;
fish_common.sorted_list_from_dict(p_dict[, ...])	根据字典的 value 进行排序，并以列表形式返回
fish_common.join_url_params(dic)	根据传入的键值对，拼接 url 后面 ? 的参数，比如 ?key1=value1&key2=value2
fish_common.has_special_char(p_str[, ...])	检查字符串是否含有指定类型字符
fish_common.find_files(path[, exts])	查找路径下的文件，返回指定类型的文件列表
fish_common.get_random_str(length[, ...])	获得指定长度，不同规则的随机字符串，可以包含数字，字母和标点符号
fish_common.get_distinct_elements(items[, key])	去除序列中的重复元素，使得剩下的元素仍然保持顺序不变，对于不可哈希的对象，需要指定 key，说明去重元素
fish_common.sort_objs_by_attr(objs, key[, ...])	对原生不支持比较操作的对象根据属性排序
fish_common.get_query_param_from_url(url)	从 url 中获取 query 参数字典

Continued on next page

Table 1 – continued from previous page

fish_common.get_sub_dict(data_dict, key_list)	从字典中提取子集
fish_common.paging(data_list[, ...])	获取分组列表数据
fish_common.camelcase_to_underline(param_dict)	将驼峰命名的参数字典键转换为下划线参数
fish_common.find_same_between_dicts(dict1, dict2)	查找两个字典中的相同点，包括键、值、项，仅支持 hashable 对象
fish_common.yaml_conf_as_dict(file_path[, ...])	读入 yaml 配置文件，返回根据配置文件内容生成的字典类型变量
fish_common.serialize_instance(obj)	对象序列化
fish_common.DeserializeInstance(obj_dict)	字典对象反序列化
fish_common.RMBConversion	人民币表示格式转换，阿拉伯数字表示的人民币和中文大写相互转换；

fish_common 包含的是最常用用的一些函数和类。

class fish_common.**DeserializeInstance**(*obj_dict*)

字典对象反序列化

Param

- obj_dict: (dict) 对象序列化字典

Returns

- obj: (object) 对象

举例如下:

```
print('--- DeserializeInstance demo ---')
temp_dict = {'user': {'name': {'last_name': 'zhang', 'first_name': 'san'},
↪'address': 'Beijing'}}
new_obj = DeserializeInstance(temp_dict)
print('last_name is: ', new_obj.user.name.last_name)
print('first_name is: ', new_obj.user.name.first_name)
print('address is: ', new_obj.user.address)
print('---')
```

执行结果:

```
--- DeserializeInstance demo ---
last_name is:  zhang
first_name is:  san
address is:  Beijing
---
```

class fish_common.**MyConfigParser**(*defaults=None*)

自定义 MyConfigParser，重写 optionxform 方法，以便读取大小写敏感的配置文件

class fish_common.**RMBConversion**

人民币表示格式转换，阿拉伯数字表示的人民币和中文大写相互转换；

举例如下:

```
print('--- RMBConversion demo ---')
chinese_amount = RMBConversion.an2cn('12345.67')
print('RMBConversion an2cn:', chinese_amount)
print('RMBConversion cn2an:', RMBConversion.cn2an(chinese_amount))
print('---')
```

执行结果:

```
--- RMBConversion demo ---
RMBConversion an2cn: 壹万贰仟叁佰肆拾伍圆陆角柒分
RMBConversion cn2an: 12345.67
---
```

static an2cn(*arabic_amount*)
将阿拉伯数字金额转换为中文大写金额表示

Param

- arabic_amount: (string) 阿拉伯数字金额

Returns

- chinese_amount: (string) 中文大写数字金额

static cn2an(*chinese_amount*)
将中文大写金额转换为阿拉伯数字金额表示

Param

- chinese_amount: (string) 中文大写数字金额

Returns

- arabic_amount: (string) 阿拉伯数字金额

class fish_common.**SingleTon**
申明一个单例类，可以作为需要单例类时候申明用的父类

Param 无

Returns 无

举例如下:

```
print('--- class singleton demo ---')
t1 = SingleTon()
t1.x = 2
print('t1.x:', t1.x)

t2 = SingleTon()

t1.x += 1

print('t1.x:', t1.x)
print('t2.x:', t2.x)
print('---')
```

执行结果:

```
--- class singleton demo ---
t1.x: 2
t1.x: 3
t2.x: 3
---
```

fish_common.**camelcase_to_underline**(*param_dict*)
将驼峰命名的参数字典键转换为下划线参数

Param

- param_dict: (dict) 请求参数字典

Returns

- temp_dict: (dict) 转换后的参数字典

举例如下:

```
print('--- transform_hump_to_underline demo---')
hump_param_dict = {'firstName': 'Python', 'Second_Name': 'san', 'right_name':
↪'name'}
underline_param_dict = transform_hump_to_underline(hump_param_dict )
print(underline_param_dict )
print('---')
```

执行结果:

```
--- transform_hump_to_underline demo---
{'first_name': 'Python', 'second_name': 'san', 'right_name': 'name'}
---
```

fish_common.**conf_as_dict**(*conf_filename*, *encoding=None*, *case_sensitive=False*)

读入 ini 配置文件，返回根据配置文件内容生成的字典类型变量;

Param

- conf_filename: (string) 需要读入的 ini 配置文件长文件名
- encoding: (string) 文件编码
- case_sensitive: (bool) 是否大小写敏感，默认为 False

Returns

- flag: (bool) 读取配置文件是否正确，正确返回 True，错误返回 False
- d: (dict) 如果读取配置文件正确返回的包含配置文件内容的字典，字典内容顺序与配置文件顺序保持一致
- count: (int) 读取到的配置文件有多少个 key 的数量

举例如下:

```
print('--- conf_as_dict demo---')
# 定义配置文件名
conf_filename = 'test_conf.ini'
# 读取配置文件
ds = conf_as_dict(conf_filename)
ds1 = conf_as_dict(conf_filename, case_sensitive=True)
# 显示是否成功，所有 dict 的内容，dict 的 key 数量
print('flag:', ds[0])
print('dict:', ds[1])
print('length:', ds[2])

d = ds[1]
d1 = ds1[1]

# 显示一个 section 下的所有内容
print('section show_opt:', d['show_opt'])
# 显示一个 section 下的所有内容，大小写敏感
print('section show_opt:', d1['show_opt'])
# 显示一个 section 下面的 key 的 value 内容
```

(continues on next page)

(continued from previous page)

```
print('section show_opt, key short_opt:', d['show_opt']['short_opt'])

# 读取一个复杂的section，先读出 key 中的 count 内容，再遍历每个 key 的 value
i = int(d['get_extra_rules']['erule_count'])
print('section get_extra_rules, key erule_count:', i)
for j in range(i):
    print('section get_extra_rules, key erule_type:', d['get_extra_rules']['erule_
↪'+str(j)])
print('---')
```

执行结果:

```
--- conf_as_dict demo---
flag: True
dict: (omit)
length: 7
section show_opt: {'short_opt': 'b:d:v:p:f:', 'long_opt': 'region=,prov=,mer_id=,
↪mer_short_name=,web_status='}
section show_opt: {'Short_Opt': 'b:d:v:p:f:', 'Long_Opt': 'region=,prov=,mer_id=,
↪mer_short_name=,web_status='}
section show_opt, key short_opt: b:d:v:p:f:
section get_extra_rules, key erule_count: 2
section get_extra_rules, key erule_type: extra_rule_1
section get_extra_rules, key erule_type: extra_rule_2
---
```

fish_common.**find_files**(*path*, *exts=None*)

查找路径下的文件，返回指定类型的文件列表

Param

- path: (string) 查找路径
- exts: (list) 文件类型列表，默认为空

Returns

- files_list: (list) 文件列表

举例如下:

```
print('--- find_files demo ---')
path1 = '/root/fishbase_issue'
all_files = find_files(path1)
print(all_files)
exts_files = find_files(path1, exts=['.png', '.py'])
print(exts_files)
print('---')
```

执行结果:

```
--- find_files demo ---
['/root/fishbase_issue/test.png', '/root/fishbase_issue/head.jpg','/root/fishbase_
↪issue/py/man.png'
['/root/fishbase_issue/test.png', '/root/fishbase_issue/py/man.png']
---
```

fish_common.**find_same_between_dicts**(*dict1*, *dict2*)

查找两个字典中的相同点，包括键、值、项，仅支持 hashable 对象

Param

- dict1: (dict) 比较的字典 1
- dict2: (dict) 比较的字典 2

Returns

- dup_info: (namedtuple) 返回两个字典中相同的信息组成的具名元组

举例如下:

```
print('--- find_same_between_dicts demo---')
dict1 = {'x':1, 'y':2, 'z':3}
dict2 = {'w':10, 'x':1, 'y':2}
res = find_same_between_dicts(dict1, dict2)
print(res.item)
print(res.key)
print(res.value)
print('---')
```

执行结果:

```
--- find_same_between_dicts demo---
set([('x', 1)])
{'x', 'y'}
{1}
---
```

fish_common.**get_distinct_elements**(*items*, *key=None*)

去除序列中的重复元素，使得剩下的元素仍然保持顺序不变，对于不可哈希的对象，需要指定 key，说明去重元素

Param

- items: (list) 需要去重的列表
- key: (hook函数) 指定一个函数，用来将序列中的元素转换成可哈希类型

Returns

- result: (generator) 去重后的结果的生成器

举例如下:

```
print('--- remove_duplicate_elements demo---')
list_demo = remove_duplicate_elements([1, 5, 2, 1, 9, 1, 5, 10])
print(list(list_demo))
list2 = [{'x': 1, 'y': 2}, {'x': 1, 'y': 3}, {'x': 1, 'y': 2}, {'x': 2, 'y': 4}]
dict_demo1 = remove_duplicate_elements(list2, key=lambda d: (d['x'], d['y']))
print(list(dict_demo1))
dict_demo2 = remove_duplicate_elements(list2, key=lambda d: d['x'])
print(list(dict_demo2))
dict_demo3 = remove_duplicate_elements(list2, key=lambda d: d['y'])
print(list(dict_demo3))
print('---')
```

执行结果:

```
--- remove_duplicate_elements demo---
[1, 5, 2, 9, 10]
[{'x': 1, 'y': 2}, {'x': 1, 'y': 3}, {'x': 2, 'y': 4}]
```

(continues on next page)

(continued from previous page)

```
[{'x': 1, 'y': 2}, {'x': 2, 'y': 4}]
[{'x': 1, 'y': 2}, {'x': 1, 'y': 3}, {'x': 2, 'y': 4}]
---
```

fish_common.**get_query_param_from_url**(*url*)

从 url 中获取 query 参数字典

Param

- url: (string) 需要获取参数字典的 url

Returns

- query_dict: (dict) query 参数的有序字典，字典的值为 query 值组成的列表

举例如下:

```
print('--- get_query_param_from_url demo---')
url = 'http://localhost:8811/mytest?page_number=1&page_size=10'
query_dict = get_query_param_from_url(url)
print(query_dict['page_size'])
print('---')
```

执行结果:

```
--- get_query_param_from_url demo---
['10']
---
```

fish_common.**get_random_str**(*length*, *letters=True*, *digits=False*, *punctuation=False*)

获得指定长度，不同规则的随机字符串，可以包含数字，字母和标点符号

Param

- length: (int) 随机字符串的长度
- letters: (bool) 随机字符串是否包含字母，默认包含
- digits: (bool) 随机字符串是否包含数字，默认不包含
- punctuation: (bool) 随机字符串是否包含特殊标点符号，默认不包含

Returns

- random_str: (string) 指定规则的随机字符串

举例如下:

```
print('--- get_random_str demo---')
print(get_random_str(6))
print(get_random_str(6, digits=True))
print(get_random_str(12, punctuation=True))
print(get_random_str(6, letters=False, digits=True))
print(get_random_str(12, letters=False, digits=True, punctuation=True))
print('---')
```

执行结果:

```
--- get_random_str demo---
nRBDHf
jXG5wR
```

(continues on next page)

(continued from previous page)

```
)I;rz{ob&Clg
427681
*"4$0^`2}%9{
---
```

fish_common.**get_sub_dict**(*data_dict*, *key_list*, *default_value='default_value'*)

从字典中提取子集

Param

- data_dict: (dict) 需要提取子集的字典
- key_list: (list) 需要获取子集的键列表
- default_value: (string) 当键不存在时的默认值，默认为 default_value

Returns

- sub_dict: (dict) 子集字典

举例如下:

```
print('--- get_sub_dict demo---')
dict1 = {'a': 1, 'b': 2, 'list1': [1,2,3]}
list1 = ['a', 'list1', 'no_key']
print(get_sub_dict(dict1, list1))
print(get_sub_dict(dict1, list1, default_value='new default'))
print('---')
```

执行结果:

```
--- get_sub_dict demo---
{'a': 1, 'list1': [1, 2, 3], 'no_key': 'default_value'}
{'a': 1, 'list1': [1, 2, 3], 'no_key': 'new default'}
---
```

fish_common.**get_time_uuid**()

获得不重复的 uuid，可以是包含时间戳的 uuid，也可以是完全随机的；基于 Python 的 uuid 类进行封装和扩展；

支持 get_time_uuid() 这样的写法，不需要参数，也可以表示生成包含时间戳的 uuid，兼容 v1.0.12 以及之前版本；

Param

- kind: (int) uuid 类型，整形常量 udTime 表示基于时间戳， udRandom 表示完全随机

Returns

- result: (string) 返回类似 66b438e3-200d-4fe3-8c9e-2bc431bb3000 的 uuid

举例如下:

```
print('--- uuid demo ---')
# 获得带时间戳的uuid
for i in range(2):
    print(get_uuid(udTime))

print('---')

# 时间戳 uuid 的简单写法，兼容之前版本
```

(continues on next page)

(continued from previous page)

```
for i in range(2):
    print(get_time_uuid())

print('---')

# 获得随机的uuid
for i in range(2):
    print(get_uuid(udRandom))

print('---')
```

执行结果:

```
--- uuid demo ---
c8aa92cc-60ef-11e8-aa87-acbf52d15413
c8ab7194-60ef-11e8-b7bd-acbf52d15413
---
c8ab7368-60ef-11e8-996c-acbf52d15413
c8ab741e-60ef-11e8-959d-acbf52d15413
---
8e108777-26a1-42d6-9c4c-a0c029423eb0
8175a81a-f346-46af-9659-077ad52e3e8f
---
```

fish_common.**get_uuid**(*kind*)

获得不重复的 uuid，可以是包含时间戳的 uuid，也可以是完全随机的；基于 Python 的 uuid 类进行封装和扩展;

支持 get_time_uuid() 这样的写法，不需要参数，也可以表示生成包含时间戳的 uuid，兼容 v1.0.12 以及之前版本;

Param

- kind: (int) uuid 类型，整形常量 udTime 表示基于时间戳， udRandom 表示完全随机

Returns

- result: (string) 返回类似 66b438e3-200d-4fe3-8c9e-2bc431bb3000 的 uuid

举例如下:

```
print('--- uuid demo ---')
# 获得带时间戳的uuid
for i in range(2):
    print(get_uuid(udTime))

print('---')

# 时间戳 uuid 的简单写法，兼容之前版本
for i in range(2):
    print(get_time_uuid())

print('---')

# 获得随机的uuid
for i in range(2):
    print(get_uuid(udRandom))

print('---')
```

执行结果:

```
--- uuid demo ---
c8aa92cc-60ef-11e8-aa87-acbf52d15413
c8ab7194-60ef-11e8-b7bd-acbf52d15413
---
c8ab7368-60ef-11e8-996c-acbf52d15413
c8ab741e-60ef-11e8-959d-acbf52d15413
---
8e108777-26a1-42d6-9c4c-a0c029423eb0
8175a81a-f346-46af-9659-077ad52e3e8f
---
```

fish_common.**has_space_element**(*source*)

判断对象中的元素，如果存在 None 或空字符串，则返回 True, 否则返回 False, 支持字典、列表和元组

Param

- source: (list, set, dict) 需要检查的对象

Returns

- result: (bool) 存在 None 或空字符串或空格字符串返回 True， 否则返回 False

举例如下:

```
print('--- has_space_element demo---')
print(has_space_element([1, 2, 'test_str']))
print(has_space_element([0, 2]))
print(has_space_element([1, 2, None]))
print(has_space_element((1, [1, 2], 3, '')))
print(has_space_element({'a': 1, 'b': 0}))
print(has_space_element({'a': 1, 'b': []}))
print('---')
```

执行结果:

```
--- has_space_element demo---
False
False
True
True
False
True
---
```

fish_common.**has_special_char**(*p_str*, *check_style=10021*)

检查字符串是否含有指定类型字符

Param

- p_str: (string) 需要判断的字符串
- check_style: (string) 需要判断的字符类型，默认为 charChinese (编码仅支持utf-8), 支持 charNum，该参数向后兼容

Returns

- True 含有指定类型字符
- False 不含有指定类型字符

举例如下:

```
print('--- has_special_char demo ---')
p_str1 = 'meiyouzhongwen'
non_chinese_result = has_special_char(p_str1, check_style=charChinese)
print(non_chinese_result)

p_str2 = u'有zhongwen'
chinese_result = has_special_char(p_str2, check_style=charChinese)
print(chinese_result)

p_str3 = 'nonnumberstring'
non_number_result = has_special_char(p_str3, check_style=charNum)
print(non_number_result)

p_str4 = 'number123'
number_result = has_special_char(p_str4, check_style=charNum)
print(number_result)
print('---')
```

执行结果:

```
--- has_special_char demo ---
False
True
False
True
---
```

fish_common.**if_json_contain**(*left_json*, *right_json*, *op='strict'*)

判断一个 json 是否包含另外一个 json 的 key，并且 value 相等;

Param

- left_json: (dict) 需要判断的 json，我们称之为 left
- right_json: (dict) 需要判断的 json，我们称之为 right，目前是判断 left 是否包含在 right 中
- op: (string) 判断操作符，目前只有一种，默认为 strict，向后兼容

Returns

- result: (bool) right json 包含 left json 的 key，并且 value 一样，返回 True，否则都返回 False

举例如下:

```
print('--- json contain demo ---')
json1 = {"id": "0001"}
json2 = {"id": "0001", "value": "File"}
print(if_json_contain(json1, json2))
print('---')
```

执行结果:

```
--- json contain demo ---
True
---
```

fish_common.**join_url_params**(*dic*)

根据传入的键值对，拼接 url 后面 ? 的参数，比如 ?key1=value1&key2=value2

Param

- dic: (dict) 参数键值对

Returns

- result: (string) 拼接好的参数

举例如下:

```
print('--- splice_url_params demo ---')
dic1 = {'key1': 'value1', 'key2': 'value2'}
print(splice_url_params(dic1))
print('---')
```

执行结果:

```
--- splice_url_params demo ---
?key1=value1&key2=value2
---
```

fish_common.**paging**(*data_list*, *group_number=1*, *group_size=10*)

获取分组列表数据

Param

- data_list: (list) 需要获取分组的数据列表
- group_number: (int) 分组信息，默认为 1
- group_size: (int) 分组大小，默认为 10

Returns

- group_data: (list) 分组数据

举例如下:

```
print('--- paging demo---')
all_records = [1, 2, 3, 4, 5]
print(get_group_list_data(all_records))

all_records1 = list(range(100))
print(get_group_list_data(all_records1, group_number=5, group_size=15))
print(get_group_list_data(all_records1, group_number=7, group_size=15))
print('---')
```

执行结果:

```
--- paging demo---
[1, 2, 3, 4, 5]
[60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74]
[90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
---
```

fish_common.**serialize_instance**(*obj*)

对象序列化

Param

- obj: (object) 对象实例

Returns

- obj_dict: (dict) 对象序列化字典

举例如下:

```
print('--- serialize_instance demo ---')
# 定义两个对象
class Obj(object):
    def __init__(self, a, b):
        self.a = a
        self.b = b

class ObjB(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

# 对象序列化
b = ObjB('string', [item for item in range(10)])
obj_ = Obj(1, b)
print(serialize_instance(obj_))
print('---')
```

执行结果:

```
--- serialize_instance demo ---
{'__classname__': 'Obj', 'a': 1,
'b': {'__classname__': 'ObjB', 'x': 'string', 'y': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]}
↪}
---
```

fish_common.**sort_objs_by_attr**(*objs*, *key*, *reverse=False*)

对原生不支持比较操作的对象根据属性排序

Param

- objs: (list) 需要排序的对象列表
- key: (string) 需要进行排序的对象属性
- reverse: (bool) 排序结果是否进行反转，默认为 False，不进行反转

Returns

- result: (list) 排序后的对象列表

举例如下:

```
print('--- sorted_objs_by_attr demo---')


class User(object):
    def __init__(self, user_id):
        self.user_id = user_id


users = [User(23), User(3), User(99)]
result = sorted_objs_by_attr(users, key='user_id')
reverse_result = sorted_objs_by_attr(users, key='user_id', reverse=True)
print([item.user_id for item in result])
print([item.user_id for item in reverse_result])
print('---')
```

执行结果:

```
--- sorted_objs_by_attr demo---
[3, 23, 99]
[99, 23, 3]
---
```

fish_common.**sorted_list_from_dict**(*p_dict*, *order=10011*)

根据字典的 value 进行排序，并以列表形式返回

Param

- p_dict: (dict) 需要排序的字典
- order: (int) 排序规则，odASC 升序，odDES 降序，默认为升序

Returns

- o_list: (list) 排序后的 list

举例如下:

```
print('--- sorted_list_from_dict demo ---')
# 定义待处理字典
dict1 = {'a_key': 'a_value', '1_key': '1_value', 'A_key': 'A_value', 'z_key': 'z_
↪value'}
print(dict1)
# 升序结果
list1 = sorted_list_from_dict(dict1, odASC)
print('ascending order result is:', list1)
# 降序结果
list1 = sorted_list_from_dict(dict1, odDES)
print('descending order result is:', list1)
print('---')
```

执行结果:

```
--- sorted_list_from_dict demo ---
{'a_key': 'a_value', 'A_key': 'A_value', '1_key': '1_value', 'z_key': 'z_value'}
ascending order result is: ['1_value', 'A_value', 'a_value', 'z_value']
descending order result is: ['z_value', 'a_value', 'A_value', '1_value']
---
```

fish_common.**yaml_conf_as_dict**(*file_path*, *encoding=None*)

读入 yaml 配置文件，返回根据配置文件内容生成的字典类型变量

Param

- file_path: (string) 需要读入的 yaml 配置文件长文件名
- encoding: (string) 文件编码
- msg: (string) 读取配置信息

Returns

- flag: (bool) 读取配置文件是否正确，正确返回 True，错误返回 False
- d: (dict) 如果读取配置文件正确返回的包含配置文件内容的字典，字典内容顺序与配置文件顺序保持一致

举例如下:

```
print('--- yaml_conf_as_dict demo---')
# 定义配置文件名
conf_filename = 'test_conf.yaml'
# 读取配置文件
ds = yaml_conf_as_dict(conf_filename, encoding='utf-8')
# 显示是否成功，所有 dict 的内容，dict 的 key 数量
print('flag:', ds[0])
print('dict length:', len(ds[1]))
print('msg:', len(ds[1]))
print('conf info: ', ds[1].get('tree'))
print('---')
```

执行结果:

```
--- yaml_conf_as_dict demo---
flag: True
dict length: 2
msg: Success
conf info:  ['README.md', 'requirements.txt', {'hellopackage': ['__init__.py']},
{'test': ['__init__.py']}, {'doc': ['doc.rst']}]
---
```

2.6.2 fish_crypt 加密数据函数包

fish_crypt.FishMD5.string(s[, salt])	获取一个字符串的 MD5 值
fish_crypt.FishMD5.file(filename)	获取一个文件的 MD5 值
fish_crypt.FishMD5.big_file(filename)	获取一个大文件的 MD5 值
fish_crypt.FishMD5.hmac_md5(s, salt)	获取一个字符串的 使用 salt 加密的 hmac MD5 值
fish_crypt.FishSha256.hmac_sha256(secret, ...)	获取一个字符串的在密钥 secret 加密下的 sha256 哈希值
fish_crypt.FishSha256.hashlib_sha256(message)	获取一个字符串的 sha256 哈希值
fish_crypt.FishBase64.string(s)	获取一个字符串的 base64 值
fish_crypt.FishBase64.file(filename)	获取一个文件的 base64 值
fish_crypt.FishBase64.decode(s)	获取 base64 解码结果

fish_crypt 包含的是一些加密、编码数据的函数，比如 MD5、SHA256 的计算。

原来这些方法属于 fish_common 模块, 因 fish_common 过于杂乱，故重新进行分类整理。

class fish_crypt.**FishBase64**

计算返回文件和字符串的 base64 编码字符串

举例如下:

```
print('--- FishBase64 demo ---')
print('string base64:', FishBase64.string('hello world!'))
file_path = get_abs_filename_with_sub_path('test_conf', 'test_conf.ini')[1])
print('file base64:', FishBase64.file(file_path)
print('decode base64:', Base64.decode(b'aGVsbG8gd29ybGQ='))
print('---')
```

执行结果:

```
--- FishBase64 demo ---
string base64: b'aGVsbG8gd29ybGQ='
file base64: (b'IyEvYmluL2Jhc2gKCmNkIC9yb290L3d3dy9zaW5nbGVfcWEKCm5vaHVwIC9yb2
90L2FwcC9weXRob24zNjIvYmluL2d1bmljb3JuIC1jIGd1bmljb3JuLmNvbmYgc2luZ2xlX3NlcnZlcjphcHAK
↪')
decode base64: b'hello world'
---
```

static decode(*s*)
获取 base64 解码结果

Param

- filename: (string) 需要进行 base64 编码 文件路径

Returns

- (bytes) base64 解码结果

static file(*filename*)
获取一个文件的 base64 值

Param

- filename: (string) 需要进行 base64 编码 文件路径

Returns

- (bytes) base64 编码结果

static string(*s*)
获取一个字符串的 base64 值

Param

- s: (string) 需要进行 base64 编码 的字符串

Returns

- (bytes) base64 编码结果

class fish_crypt.**FishMD5**
计算普通字符串和一般的文件，对于大文件采取逐步读入的方式，也可以快速计算；基于 Python 的 hashlib.md5() 进行封装和扩展；

举例如下:

```
print('--- md5 demo ---')
print('string md5:', GetMD5.string('hello world!'))
file_path = get_abs_filename_with_sub_path('test_conf', 'test_conf.ini')[1])
print('file md5:', GetMD5.file(file_path)
big_file_path = get_abs_filename_with_sub_path('test_conf', 'test_conf.ini')[1])
print('big file md5:', GetMD5.big_file(big_file_path)
print('string hmac_md5:', GetMD5.hmac_md5('hello world!', 'salt'))
print('---')
```

执行结果:

```
--- md5 demo ---
string md5: fc3ff98e8c6a0d3087d515c0473f8677
file md5: fb7528c9778b2377e30b0f7e4c26fef0
big file md5: fb7528c9778b2377e30b0f7e4c26fef0
```

(continues on next page)

(continued from previous page)

```
string hmac_md5: 191f82804523bfdafe0188bbbddd6587
---
```

static big_file(*filename*)
获取一个大文件的 MD5 值

Param

- filename: (string) 需要进行 hash 的大文件路径

Returns

- result: (string) 32位小写 MD5 值

static file(*filename*)
获取一个文件的 MD5 值

Param

- filename: (string) 需要进行 hash 的文件名

Returns

- result: (string) 32位小写 MD5 值

static hmac_md5(*s*, *salt*)
获取一个字符串的 使用 salt 加密的 hmac MD5 值

Param

- s: (string) 需要进行 hash 的字符串
- salt: (string) 随机字符串

Returns

- result: (string) 32位小写 MD5 值

static string(*s*, *salt=None*)
获取一个字符串的 MD5 值

Param

- s: (string) 需要进行 hash 的字符串
- salt: (string) 随机字符串，默认为 None

Returns

- result: (string) 32 位小写 MD5 值

class fish_crypt.**FishSha256**
计算字符串和密钥的 sha256 算法哈希值

举例如下:

```
print('--- GetSha256 demo ---')
# 定义哈希字符串
message = 'Hello HMAC'
# 定义密钥
secret = '12345678'
print('hmac_sha256:', GetSha256.hmac_sha256(secret, message))
print('hashlib_sha256:', GetSha256.hashlib_sha256(message))
print('---')
```

执行结果:

```
--- GetSha256 demo ---
hmac_sha256: 5eb8bdabdaa43f61fb220473028e49d40728444b4322f3093decd9a356afd18f
hashlib_sha256: 4a1601381dfb85d6e713853a414f6b43daa76a82956911108512202f5a1c0ce4
---
```

static hashlib_sha256(*message*)
获取一个字符串的 sha256 哈希值

Param

- message: (string) 需要进行哈希的字符串

Returns

- hashed_str: sha256 算法哈希值

static hmac_sha256(*secret*, *message*)
获取一个字符串的在密钥 secret 加密下的 sha256 哈希值

Param

- secret: (string) 哈希算法的密钥
- message: (string) 需要进行哈希的字符串

Returns

- hashed_str: sha256 算法哈希值

2.6.3 fish_csv csv 函数包

fish_csv.csv2list(csv_filename[, deli, ...])	将指定的 csv 文件转换为 list 返回;
fish_csv.list2csv(data_list[, csv_filename])	将字典写入到指定的 csv 文件，并返回文件的长文件名;
fish_csv.csv2dict(csv_filename[, deli, ...])	将指定的 csv 文件转换为 list 返回;
fish_csv.dict2csv(data_dict[, csv_filename, ...])	将字典写入到指定的 csv 文件，并返回文件的长文件名;

fish_csv.**csv2dict**(*csv_filename*, *deli=','*, *encoding=None*, *key_is_header=False*)
将指定的 csv 文件转换为 list 返回;

Param

- csv_filename: (string) csv 文件的长文件名
- deli: (string) csv 文件分隔符，默认为逗号
- del_blank_row: (string) 是否要删除空行，默认为删除
- encode: (string) 文件编码

Returns

- csv_data: (dict) 读取后的数据

举例如下:

```
from fishbase.fish_file import *
from fishbase.fish_csv import *
```

(continues on next page)

(continued from previous page)

```
def test_csv2dict():
    csv_filename = get_abs_filename_with_sub_path('csv', 'test_csv.csv')[1]
    print(csv_filename)
    csv_dict = csv2dict(csv_filename)
    print(csv_dict)


if __name__ == '__main__':
    test_csv2dict()
```

fish_csv.**csv2list**(*csv_filename*, *deli=','*, *del_blank_row=True*, *encoding=None*)

将指定的 csv 文件转换为 list 返回;

Param

- csv_filename: (string) csv 文件的长文件名
- deli: (string) csv 文件分隔符，默认为逗号
- del_blank_row: (string) 是否要删除空行，默认为删除
- encode: (string) 文件编码

Returns

- csv_list: (list) 转换后的 list

举例如下:

```
from fishbase.fish_file import *
from fishbase.fish_csv import *

def test_csv():
    csv_filename = get_abs_filename_with_sub_path('csv', 'test_csv.csv')[1]
    print(csv_filename)
    csv_list = csv2list(csv_filename)
    print(csv_list)


if __name__ == '__main__':
    test_csv()
```

fish_csv.**csv_file_to_list**(*csv_filename*, *deli=','*, *del_blank_row=True*, *encoding=None*)

将指定的 csv 文件转换为 list 返回;

Param

- csv_filename: (string) csv 文件的长文件名
- deli: (string) csv 文件分隔符，默认为逗号
- del_blank_row: (string) 是否要删除空行，默认为删除
- encode: (string) 文件编码

Returns

- csv_list: (list) 转换后的 list

举例如下:

```
from fishbase.fish_file import *
from fishbase.fish_csv import *

def test_csv():
    csv_filename = get_abs_filename_with_sub_path('csv', 'test_csv.csv')[1]
    print(csv_filename)
    csv_list = csv2list(csv_filename)
    print(csv_list)


if __name__ == '__main__':
    test_csv()
```

fish_csv.**dict2csv**(*data_dict*, *csv_filename='./dict2csv.csv'*, *key_is_header=False*)
将字典写入到指定的 csv 文件，并返回文件的长文件名;

Param

- data_dict: (dict) 需要写入 csv 的数据字典
- csv_filename: (string) csv 文件的长文件名
- key_is_header: (bool) csv 文件第一行是否全为字典 key

Returns

- csv_filename: (string) csv 文件的长文件名

举例如下:

```
from fishbase.fish_csv import *

def test_dict2csv():
    data_dict = {'a': 1, 'b': 2}
    csv_file = dict2csv(data_dict)
    print(csv_file)


if __name__ == '__main__':
    test_dict2csv()
```

fish_csv.**list2csv**(*data_list*, *csv_filename='./list2csv.csv'*)
将字典写入到指定的 csv 文件，并返回文件的长文件名;

Param

- data_list: (list) 需要写入 csv 的数据字典
- csv_filename: (string) csv 文件的长文件名

Returns

- csv_filename: (string) csv 文件的长文件名

举例如下:

```
from fishbase.fish_csv import *

def test_list2csv():
    data_list = ['a', 'b', 'c']
    csv_file = list2csv(data_list)
    print(csv_file)
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    test_list2csv()
```

2.6.4 fish_data 数据信息处理函数包，含银行卡、身份证等

fish_data.CardBin.get_checkcode(card_number_str)	计算银行卡校验位；
fish_data.CardBin.check_bankcard(card_number_str)	检查银行卡校验位是否正确；
fish_data.CardBin.get_bank_info(bankname)	银行名称，返回银行代码；
fish_data.CardBin.get_cardbin_info(bank, ...)	输入银行、借记贷记卡种类，返回有效的卡 bin；
fish_data.IdCard.get_checkcode(id_number_str)	计算身份证号码的校验位；
fish_data.IdCard.check_number(id_number)	检查身份证号码是否符合校验规则；
fish_data.IdCard.get_zone_info(area_str[, ...])	输入包含省份、城市、地区信息的内容，返回地区编号；
fish_data.IdCard.get_areanote_info(province)	输入省份代码，返回地区信息；
fish_data.IdCard.get_province_info()	获取省份代码

fish_data 包含的是一些与数据信息相关的函数，比如银行卡、身份证信息的生成和校验。

在我们进行一些开发测试、功能测试、自动化测试、压力测试等场景下，都需要模拟身份证、银行卡等信息。 fish_data 中的函数就是用在这样的场景。注意，这些函数不会生成真实的身份证和银行卡号。

class fish_data.**CardBin**

校验银行卡号、获取银行卡校验位，获取银行卡、银行信息；

举例如下:

```
print('--- CardBin demo ---')

print('get_checkcode of "439188000699010":', CardBin.get_checkcode(
↪'439188000699010'))

print('check_bankcard of "4391880006990100":', CardBin.check_number(
↪'4391880006990100'))

print('get_bank_info of "招商银行":', CardBin.get_bank_info('招商银行')

print('get_cardbin_info of "CMB", "DC":', CardBin.get_cardbin_info('CMB', 'DC'))

print('---')
```

执行结果:

```
--- CardBin demo ---
```

(continues on next page)

(continued from previous page)

```
get_checkcode of "439188000699010": 9

check_bankcard of "4391880006990100": False

get_bank_info of "招商银行": [('CMB', '招商银行')]

get_cardbin_info of "CMB", "DC": [('410062', 'CMB', 'DC', 16), ('468203', 'CMB',
↪'DC', 16), ...

---
```

classmethod check_bankcard(*card_number_str*)
检查银行卡校验位是否正确;

Param

- card_number_str: (string) 要查询的银行卡号

Returns 返回结果：(bool) True or False

举例如下:

```
from fishbase.fish_data import *

print('--- fish_data check_bankcard demo ---')

# 不能放真的卡信息，有风险
print(CardBin.check_bankcard('4391880006990100'))

print('---')
```

输出结果:

```
--- fish_data check_bankcard demo ---
False
---
```

classmethod get_bank_info(*bankname*)
银行名称，返回银行代码;

Param

- bankname: (string) 要查询的银行 名称，比如 招商银行

Returns

- 返回 银行代号bank , 银行名称 bankname，一条记录为一个 tuple (a, b)，然后组成 list 返回

举例如下:

```
from fishbase.fish_data import *

print('--- fish_data get_bank_info demo ---')

print(CardBin.get_bank_info('招商银行'))

print('---')
```

输出结果:

```
--- fish_data get_bank_info demo ---
[('CMB', '招商银行')]
---
```

classmethod get_cardbin_info(*bank*, *card_type*)
输入银行、借记贷记卡种类，返回有效的卡 bin；

Param

- bank: (string) 要查询的银行代号，比如 ICBC, CMB
- card_type: (string) 银行卡类型，比如 CC 表示信用卡

Returns

- 返回 cardbin, bank, 银行卡类型type, 银行卡长度 length，一条记录为一个 tuple (a, b, c, d)，然后组成 list 返回

举例如下:

```
from fishbase.fish_data import *

print('--- fish_data get_cardbin_info demo ---')

result = CardBin.get_cardbin_info('CMB', 'DC')
print(result)

print('---')
```

输出结果:

```
--- fish_data get_cardbin_info demo ---

[('410062', 'CMB', 'DC', 16), ('468203', 'CMB', 'DC', 16), ...
---
```

classmethod get_checkcode(*card_number_str*)
计算银行卡校验位；

Param

- card_number_str: (string) 要查询的银行卡号

Returns checkcode: (string) 银行卡的校验位

举例如下:

```
from fishbase.fish_data import *

print('--- fish_data get_checkcode demo ---')

# 不能放真的卡信息，有风险
print(CardBin.get_checkcode('439188000699010'))

print('---')
```

输出结果:

```
--- fish_data get_checkcode demo ---
9
---
```

class fish_data.**IdCard**

校验身份证号、获取身份证校验位，获取随机生成身份证号所需身份代码等函数;

举例如下:

```
print('--- IdCard demo ---')

print('get_checkcode of "32012419870101001":', IdCard.get_checkcode(
↪'32012419870101001')[1])

print('check_number of "130522198407316471":', IdCard.check_number(
↪'130522198407316471')[0])

print('get_zone_info of "北京市":', IdCard.get_zone_info(area_str='北京市')

print('get_areanote_info of "北京(11)":', IdCard.get_areanote_info('11'))

print('---')
```

执行结果:

```
--- IdCard demo ---

get_checkcode of "32012419870101001": 5

check_number of "130522198407316471": True

get_zone_info of "北京市": [('110000', '北京市')]

get_areanote_info of "北京(11)": ([('110000', '北京市'), ('110100', '北京市市辖区'), (
↪'110101', '北京市东城区'), ...

---
```

classmethod check_number(*id_number*)

检查身份证号码是否符合校验规则;

Param

- id_number: (string) 身份证号，比如 32012419870101001

Returns

- 返回类型 (tuple)，当前有一个值，第一个为 flag，以后第二个值会返回具体校验不通过的详细错误
- flag: (bool) 如果身份证号码校验通过，返回 True；如果身份证校验不通过，返回 False

举例如下:

```
from fishbase.fish_data import *

print('--- fish_data check_number demo ---')

# id number false
id1 = '320124198701010012'
print(id1, IdCard.check_number(id1)[0])

# id number true
```

(continues on next page)

(continued from previous page)

```
id2 = '130522198407316471'
print(id2, IdCard.check_number(id2)[0])

print('---')
```

输出结果:

```
--- fish_data check_number demo ---
320124198701010012 False
130522198407316471 True
---
```

classmethod get_areanote_info(*province*)
输入省份代码，返回地区信息;

Param

- province_code: (string) 省份代码 比如：11

Returns

- note_list: (list) 地区信息列表

举例如下:

```
from fishbase.fish_data import *

print('--- fish_data get_areanote_info demo ---')

print(IdCard.get_areanote_info('11'))

print('---')
```

输出结果:

```
--- fish_data get_areanote_info demo ---
[('110000', '北京市'), ('110100', '北京市市辖区'), ('110101', '北京市东城区'), ...

---
```

classmethod get_checkcode(*id_number_str*)
计算身份证号码的校验位;

Param

- id_number_str: (string) 身份证号的前17位，比如 3201241987010100

Returns

- 返回类型 (tuple)
- flag: (bool) 如果身份证号格式正确，返回 True；格式错误，返回 False
- checkcode: 计算身份证前17位的校验码

举例如下:

```
from fishbase.fish_data import *

print('--- fish_data get_checkcode demo ---')
```

(continues on next page)

(continued from previous page)

```
# id number
id1 = '32012419870101001'
print(id1, IdCard.get_checkcode(id1)[1])

# id number
id2 = '13052219840731647'
print(id2, IdCard.get_checkcode(id2)[1])

print('---')
```

输出结果:

```
--- fish_data get_checkcode demo ---
32012419870101001 5
13052219840731647 1
---
```

classmethod get_province_info()
获取省份代码

Param

Returns

- province_list: (list) 省份代码列表

举例如下:

```
from fishbase.fish_data import *

print('--- fish_data get_province_info demo ---')
print(IdCard.get_province_info())
print('---')
```

输出结果:

```
--- fish_data get_province_info demo ---
```

[('11',), ('12',), ('13',), ('14',), ('15',), ... —

classmethod get_zone_info(*area_str*, *match_type='EXACT'*, *result_type='LIST'*)
输入包含省份、城市、地区信息的内容，返回地区编号;

Param

- area_str: (string) 要查询的区域，省份、城市、地区信息，比如 北京市
- match_type: (string) 查询匹配模式，默认值 'EXACT'，表示精确匹配，可选 'FUZZY'，表示模糊查询
- result_type: (string) 返回结果数量类型，默认值 'LIST'，表示返回列表，可选 'SINGLE_STR'，返回结果的第一个地区编号字符串

Returns

- 返回类型 根据 resule_type 决定返回类型是列表或者单一字符串，列表中包含元组 比如：[('110000', '北京市')]，元组中的第一个元素是地区码，

第二个元素是对应的区域内容 结果最多返回 20 个。

举例如下:

```
from fishbase.fish_data import *

print('--- fish_data get_zone_info demo ---')

result = IdCard.get_zone_info(area_str='北京市')
print(result)

# 模糊查询
result = IdCard.get_zone_info(area_str='西安市', match_type='FUZZY')
print(result)

result0 = []
for i in result:
    result0.append(i[0])

print('---西安市---')
print(len(result0))
print(result0)

# 模糊查询, 结果返回设定 single_str
result = IdCard.get_zone_info(area_str='西安市', match_type='FUZZY', result_
↪type='SINGLE_STR')
print(result)

# 模糊查询, 结果返回设定 single_str, 西安市 和 西安 的差别
result = IdCard.get_zone_info(area_str='西安', match_type='FUZZY', result_
↪type='SINGLE_STR')
print(result)

print('---')
```

输出结果:

```
--- fish_data get_zone_info demo ---
[('110000', '北京市')]
130522198407316471 True

---西安市---
11
['610100', '610101', '610102', '610103', '610104', '610111', '610112', '610113
↪', '610114', '610115',
'610116']

610100
220403
---
```

2.6.5 fish_date 日期处理增强函数包

fish_date.get_date_range(dates[, separator])	获取某个月的日期范围，返回该月第一天和最后一天的字符串表示
fish_date.get_years([months, refer])	获取基准时月份增量的年月
fish_date.GetRandomTime	获取随机时间

Continued on next page

Table 5 – continued from previous page

fish_date.get_time_interval(start_time, end_time)	获取两个unix时间戳之间的时间间隔
fish_date.transform_unix_to_datetime(timestamp)	将 unix 时间戳转换成 datetime 类型
fish_date.transform_datetime_to_unix([dtime])	将 datetime 类型转换成 unix 时间戳
fish_date.FishDateTimeFormat	实现 datetime 和 str 之间相互转换，基于 Python 的 datetime.datetime 进行封装和扩展；

class fish_date.**FishDateTimeFormat**

实现 datetime 和 str 之间相互转换，基于 Python 的 datetime.datetime 进行封装和扩展；

举例如下:

```
print('--- FishDateTimeFormat demo ---')
datetime_obj = date(year=2018, month=11, day=23)
print(FishDateTimeFormat.strftime(datetime_obj, '%Y-%m-%d'))
date_time_str = '2018-11-23 23:17:20'
time_format = '%Y-%m-%d %H:%M:%S'
print(FishDateTimeFormat.strptime(date_time_str, time_format))
print('---')
```

执行结果:

```
--- FishDateTimeFormat demo ---
2018-11-23
2018-11-23 23:17:20
<class 'datetime.datetime'>
---
```

static strftime(*date_time=None*, *time_format=None*)

将 datetime 对象转换为 str

Param

- date_time: (obj) datetime 对象
- time_format: (sting) 日期格式字符串

Returns

- date_time_str: (string) 日期字符串

static strptime(*time_str*, *time_format*)

将 str 转换为 datetime 对象

Param

- time_str: (string) 日期字符串
- time_format: (sting) 日期格式字符串

Returns

- datetime_obj: (obj) datetime 对象

class fish_date.**GetRandomTime**

获取随机时间

举例如下:

```
print('--- GetRandomTime demo ---')
print(GetRandomTime.date_time_this_month())
print(GetRandomTime.date_time_this_year())
print('---')
```

执行结果:

```
--- Base64 demo ---
2018-07-01 12:47:20
2018-02-08 17:16:09
---
```

static date_time_this_month()
获取当前月的随机时间

Returns

- date_this_month: (datetime) 当前月份的随机时间

举例如下:

```
print('--- GetRandomTime.date_time_this_month demo ---')
print(GetRandomTime.date_time_this_month())
print('---')
```

执行结果:

```
--- GetRandomTime.date_time_this_month demo demo ---
2018-07-01 12:47:20
---
```

static date_time_this_year()
获取当前年的随机时间字符串

Returns

- date_this_year: (datetime) 当前月份的随机时间

举例如下:

```
print('--- GetRandomTime.date_time_this_year demo ---')
print(GetRandomTime.date_time_this_year())
print('---')
```

执行结果:

```
--- GetRandomTime.date_time_this_year demo demo ---
2018-02-08 17:16:09
---
```

static gen_date_by_range(*begin_date*, *end_date*, *date_format='%Y-%m-%d'*)
指定一个日期范围，随机生成区间内的某一个日期，该区间为闭区间

Param

- begin_date: (string) 范围的起始日期，字符串 yyyy-MM-dd eg. 2018-01-01
- end_date: (string) 范围的结束日期，字符串 yyyy-MM-dd eg. 2018-12-31
- date_format: 返回的日期格式，字符串：默认格式yyyyMMdd default: "%Y%m%d"

Returns

- date_str 日期区间内的一个指定格式的合法的随机日期

举例如下:

```
print('--- GetRandomTime.gen_date_by_range demo ---')
print(GetRandomTime.gen_date_by_range("2010-01-01","2010-12-31"))
print('---')
```

执行结果:

```
--- GetRandomTime.gen_date_by_range demo ---
20100124
---
```

static gen_date_by_year(*year*)

获取当前年的随机时间字符串

Param

- year: (string) 长度为 4 位的年份字符串

Returns

- date_str: (string) 传入年份的随机合法的日期

举例如下:

```
print('--- GetRandomTime.gen_date_by_year demo ---')
print(GetRandomTime.gen_date_by_year("2010"))
print('---')
```

执行结果:

```
--- GetRandomTime.gen_date_by_year demo ---
20100505
---
```

fish_date.**get_date_range**(*dates*, *separator='-'*)

获取某个月的日期范围，返回该月第一天和最后一天的字符串表示

Param

- dates: (string 或者 datetime obj) 月份信息
- separator: (string) 分隔符，默认为 '-'

Returns

- first_day: (string) 该月份的第一天
- last_day: (string) 该月份的最后一天

举例如下:

```
print('--- get_date_range demo ---')
now_time = datetime.now()
print(get_date_range(now_time))
print(get_date_range('201802',separator='/'))
print('---')
```

执行结果:

```
--- get_years demo ---
('2018-06-1', '2018-06-30')
('2018/02/1', '2018/02/28')
---
```

fish_date.**get_time_interval**(*start_time*, *end_time*)
获取两个unix时间戳之间的时间间隔

Param

- start_time: (int) 开始时间，unix 时间戳
- end_time: (int) 结束时间，unix 时间戳

Returns

- interval_dict: (dict) 时间间隔字典

举例如下:

```
print('--- get_time_interval demo ---')
import time
start = int(time.time())
end = start - 98908
print(get_time_interval(end, start))
print('---')
```

执行结果:

```
--- get_time_interval demo ---
{'days': 1, 'hours': 3, 'minutes': 28, 'seconds': 28}
---
```

fish_date.**get_years**(*months=0*, *refer=None*)
获取基准时月份增量的年月

Param

- months: (int) 月份增量，正数为往后年月，整数为往前年月
- refer: (datetime obj) datetime 对象，或者有 month 和 year 属性的实例，默认为当前时间

Returns

- result: (string) 年月字符串

举例如下:

```
print('--- get_years demo ---')
print(get_years(-5))
print(get_years(7, datetime.now()))
print('---')
```

执行结果:

```
--- get_years demo ---
201801
201901
---
```

fish_date.**transform_datetime_to_unix**(*dtime=None*)
将 datetime 类型转换成 unix 时间戳

Param

- dtime: (datetime) datetime 类型实例,默认为当前时间

Returns

- data_type: (datetime) datetime 类型实例

举例如下:

```
print('--- transform_datetime_to_unix demo ---')
dtime = datetime.datetime.now()
ans_time = transform_datetime_to_unix(dtime)
print(ans_time)
print('---')
```

执行结果:

```
--- transform_datetime_to_unix demo ---
1535108620.0
---
```

fish_date.**transform_unix_to_datetime**(*timestamp*)
将 unix 时间戳转换成 datetime 类型

Param

- timestamp: (int) unix 时间戳

Returns

- data_type: (datetime) datetime 类型实例

举例如下:

```
print('--- transform_unix_to_datetime demo ---')
import time
timestamp = int(time.time())
date_type = transform_unix_to_datetime(timestamp)
print(type(date_type))
print(date_type)

print('---')
```

执行结果:

```
--- transform_unix_to_datetime demo ---
<class 'datetime.datetime'>
2018-08-22 19:48:03
---
```

2.6.6 fish_file 文件处理函数包

fish_file.check_sub_path_create(sub_path)	检查当前路径下的某个子路径是否存在, 不存在则创建;

Continued on next page

Table 6 – continued from previous page

fish_file.get_abs_filename_with_sub_path(sub_path, filename)	生成当前路径下一级路径某文件的完整文件名;
fish_file.get_file_encoding(file_path)	获取给定文件的编码;

fish_file 包含的是文件、路径处理相关的函数。

各类相对绝对文件的路径处理等都是开发时候经常需要处理的问题，fish_file 中的函数试图简化这些操作。

fish_file.**check_sub_path_create**(*sub_path*)

检查当前路径下的某个子路径是否存在, 不存在则创建;

Param

- sub_path: (string) 下一级的某路径名称

Returns

- 返回类型 (tuple)，有两个值
- True: 路径存在，False: 不需要创建
- False: 路径不存在，True: 创建成功

举例如下:

```
print('--- check_sub_path_create demo ---')
# 定义子路径名称
sub_path = 'demo_sub_dir'
# 检查当前路径下的一个子路径是否存在，不存在则创建
print('check sub path:', sub_path)
result = check_sub_path_create(sub_path)
print(result)
print('---')
```

输出结果:

```
--- check_sub_path_create demo ---
check sub path: demo_sub_dir
(True, False)
---
```

fish_file.**get_abs_filename_with_sub_path**(*sub_path*, *filename*)

生成当前路径下一级路径某文件的完整文件名;

Param

- sub_path: (string) 下一级的某路径名称
- filename: (string) 下一级路径的某个文件名

Returns

- 返回类型 (tuple)，有两个值，第一个为 flag，第二个为文件名，说明见下
- flag: (bool) 如果文件存在，返回 True，文件不存在，返回 False
- abs_filename: (string) 指定 filename 的包含路径的长文件名

举例如下:

```
print('--- get_abs_filename_with_sub_path demo ---')
# define sub dir
path_name = 'sub_dir'
```

(continues on next page)

(continued from previous page)

```
# define not exists file
filename = 'test_file.txt'

abs_filename = get_abs_filename_with_sub_path(path_name, filename)
# return False and abs filename
print(abs_filename)

# define exists file
filename = 'demo.txt'
abs_filename = get_abs_filename_with_sub_path(path_name, filename)
# return True and abs filename
print(abs_filename)
print('---')
```

输出结果:

```
--- get_abs_filename_with_sub_path demo ---
(False, '/Users/****/Documents/dev_python/fishbase/demo/sub_dir/test_file.txt')
(True, '/Users/****/Documents/dev_python/fishbase/demo/sub_dir/demo.txt')
---
```

fish_file.**get_file_encoding**(*file_path*)

获取给定文件的编码;

Param

- file_path: (string) 文件的完整路径

Returns

- file_encoding (string)，文件编码

举例如下:

```
print('--- get_file_encoding demo ---')
result = get_file_encoding(__file__)
print(result)
print('---')
```

输出结果:

```
--- get_file_encoding demo ---
utf-8
---
```

2.6.7 fish_logger 日志记录函数包

fish_logger 包含的是日志处理相关的函数。

通过 set_log_file() 可以方便的进行分卷的日志文件记录。

fish_logger.**set_log_file**(*local_file=None*)

设置日志记录，按照每天一个文件，记录包括 info 以及以上级别的内容； 日志格式采取日志文件名直接加上日期，比如 fish_test.log.2018-05-27

Param

- local_fie: (string) 日志文件名

Returns 无

举例如下:

```
from fishbase.fish_logger import *
from fishbase.fish_file import *

log_abs_filename = get_abs_filename_with_sub_path('log', 'fish_test.log')[1]

set_log_file(log_abs_filename)

logger.info('test fish base log')
logger.warn('test fish base log')
logger.error('test fish base log')

print('log ok')
```

fish_logger.**set_log_stdout**()
设置输出到标准输出中

Param 无

Returns 无

举例如下:

```
from fishbase.fish_logger import *

set_log_stdout()

logger.info('test fish base log')
logger.warn('test fish base log')
logger.error('test fish base log')

print('log ok')
```

2.6.8 fish_project project 函数包

fish_project.init_project_by_yml([...])	通过配置文件初始化一个 project

fish_project 用来根据配置文件创建项目工程。

fish_project.**init_project_by_yml**(*project_config=None*, *dist=None*)
通过配置文件初始化一个 project

Param

- project_config: (string) 用来生成 project 的配置文件
- dist: (string) project 位置

举例如下:

```
print('--- init_project_by_yml demo ---')
# define yml string
package_yml = '''
project: hellopackage
tree:
```

(continues on next page)

(continued from previous page)

```
    - README.md
    - requirements.txt
    - setup.py
    - MANIFEST.in
    - hellopackage: # project name
        - __init__.py
    - test: # unittest file
        - __init__.py
    - demo: # usage demo
        - __init__.py
    - doc: # documents
'''
# init project by yml
init_project_by_yml(package_yml, '.')
print(os.listdir('./hellopackage'))
print('---')
```

输出结果:

```
--- init_project_by_yml demo ---
['demo', 'requirements.txt', 'test', 'MANIFEST.in', 'hellopackage', 'README.md',
↪'setup.py', 'doc']
---
```

2.6.9 fish_system 系统相关函数包

fish_system.get_platform()	返回当前程序运行的操作系统名称, 基于 sys.platform() 进行封装;

fish_system 包含的是一些系统相关的函数和类。

fish_system.**get_platform**()

返回当前程序运行的操作系统名称, 基于 sys.platform() 进行封装;

Param

- 无

Returns

- platform: (string) 返回 linux, osx, win 或者其他

举例如下:

```
print('current os:', get_platform())
```

执行结果:

```
current os: osx
```

2.6.10 fish_random 随机数据生成函数包

fish_random.gen_random_address(zone)	通过省份行政区划代码，返回该省份的随机地址
fish_random.get_random_areanote(zone)	省份行政区划代码，返回下辖的随机地区名称
fish_random.gen_random_bank_card([...])	通过指定的银行名称，随机生成该银行的卡号
fish_random.gen_random_company_name()	随机生成一个公司名称
fish_random.gen_random_float(minimum, maximum)	指定一个浮点数范围，随机生成并返回区间内的一个浮点数，区间为闭区间 受限于 random.random 精度限制，支持最大 15 位精度
fish_random.gen_random_id_card([zone, ...])	根据指定的省份编号、性别或年龄，随机生成一个身份证号
fish_random.gen_random_mobile()	随机生成一个手机号
fish_random.gen_random_name([family_name, ...])	指定姓氏、性别、长度，返回随机人名，也可不指定生成随机人名
fish_random.gen_random_str(min_length, ...)	指定一个前后缀、字符串长度以及字符串包含字符类型，返回随机生成带有前后缀及指定长度的字符串

fish_random 包含的是一些生成随机值的函数。

fish_random.**gen_random_address**(*zone*)
通过省份行政区划代码，返回该省份的随机地址

Param

- zone: (string) 省份行政区划代码 比如 '310000'

Returns

- random_addr: (string) 省份下辖随机地区名称

举例如下:

```
print('--- gen_address demo ---')
print(gen_address('310000'))
print('---')
```

输出结果:

```
--- gen_address demo ---
上海市卢湾区陵县支街918号
---
```

fish_random.**gen_random_bank_card**(*bank_name=None*, *card_type=None*)
通过指定的银行名称，随机生成该银行的卡号

Param

- bank_name: (string) 银行名称 eg. 中国银行
- card_type：(string) 卡种类，可选 CC(信用卡)、DC(借记卡)

Returns

- random_bank_card: (string) 随机生成的银行卡卡号

举例如下:

```
print('--- gen_random_bank_card demo ---')
print(gen_bank_card())
print(gen_bank_card('中国银行', 'CC'))
```

(continues on next page)

(continued from previous page)

```
print(gen_bank_card('中国银行', 'DC'))
print('---')
```

输出结果:

```
--- gen_random_bank_card demo ---
6282689914390956
6259073791134721
6212836989522229131
---
```

fish_random.**gen_random_company_name**()
随机生成一个公司名称

Returns

- company_name: (string) 银行名称

举例如下:

```
print('--- gen_random_company_name demo ---')
print(gen_random_company_name())
print('---')
```

输出结果:

```
--- gen_random_company_name demo ---
上海大升旅游质询有限责任公司
---
```

fish_random.**gen_random_float**(*minimum*, *maximum*, *decimals=2*)
指定一个浮点数范围，随机生成并返回区间内的一个浮点数，区间为闭区间 受限于 random.random 精度限制，支持最大 15 位精度

Param

- minimum: (float) 浮点数最小取值
- maximum: (float) 浮点数最大取值
- decimals: (int) 小数位数，默认为 2 位

Returns

- random_float: (float) 随机浮点数

举例如下:

```
print('--- gen_random_float demo ---')
print(gen_random_float(1.0, 9.0))
print(gen_random_float(1.0, 9.0, decimals=10))
print(gen_random_float(1.0, 9.0, decimals=20))
print('---')
```

执行结果:

```
--- gen_random_float demo ---
6.08
6.8187342239
```

(continues on next page)

(continued from previous page)

```
2.137902497554043
---
```

fish_random.**gen_random_id_card**(*zone=None*, *gender=None*, *age=None*, *result_type='SINGLE_STR'*)

根据指定的省份编号、性别或年龄，随机生成一个身份证号

Param

- zone: (string) 省份编号 eg. 310000, 默认 None: 随机
- gender：(string) 性别 "01" 男性， "00" 女性, 默认 None: 随机
- age：(int) 年龄 默认 None：随机 身份证最早出生年份为 1970
- result_type: (string) 返回结果数量类型，默认值 'SINGLE_STR'，表示随机返回一个身份证号，可选 'LIST'，返回一个随机身份证列表

Returns

- id_num_list: (list) 随机生成的身份证号组成的列表

举例如下:

```
print('--- gen_random_id_card demo ---')
print(gen_id('310000'))
print(gen_id('310000', age=100))
print(gen_id('310000', age=30, gender='00'))
print(gen_id(age=30, gender='01', result_type='LIST'))
print('---')
```

输出结果:

```
--- gen_random_id_card demo ---
['310109198610243547']
['310101197006245479']
['310101198808249062']
['441229198805145278', '440507198812196011', '441622198805222074',
↪'441721198801046033', ...
---
```

fish_random.**gen_random_mobile**()

随机生成一个手机号

Returns

- str: (string) 手机号

举例如下:

```
print('--- gen_random_mobile demo ---')
print(gen_random_mobile())
print(gen_random_mobile())
print('---')
```

执行结果:

```
--- gen_random_mobile demo ---
16706146773
14402633925
---
```

fish_random.**gen_random_name**(*family_name=None*, *gender=None*, *length=None*)
指定姓氏、性别、长度，返回随机人名，也可不指定生成随机人名

Param

- family_name: (string) 姓
- gender: (string) 性别 “01” 男性， “00” 女性, 默认 None: 随机
- length: (int) 大于等于 2 小于等于 10 的整数, 默认 None: 随机 2 或者 3

Returns

- full_name: (string) 随机人名

举例如下:

```
print('--- gen_random_name demo ---')
print(gen_name())
print(gen_name("赵","01", 3))
print('---')
```

执行结果:

```
--- gen_random_name demo ---
师艺
赵群腾
---
```

fish_random.**gen_random_str**(*min_length*, *max_length*, *prefix=None*, *suffix=None*, *has_letter=True*, *has_digit=False*, *has_punctuation=False*)
指定一个前后缀、字符串长度以及字符串包含字符类型，返回随机生成带有前后缀及指定长度的字符串

Param

- min_length: (int) 字符串最小长度
- max_length: (int) 字符串最小长度
- prefix: (string) 字符串前缀
- suffix: (string) 字符串后缀
- has_letter: (bool) 字符串时候包含字母，默认为 True
- has_digit: (bool) 字符串是否包含数字，默认为 False
- has_punctuation: (bool) 字符串是否包含标点符号，默认为 False

Returns

- random_str: (string) 指定规则的随机字符串

举例如下:

```
print('--- gen_random_str demo ---')
print(gen_random_str(5, 7))
print(gen_random_str(5, 7, prefix='FISHBASE_'))
print(gen_random_str(5, 7, prefix='FISHBASE_', suffix='.py'))
print(gen_random_str(5, 7, has_digit=True, has_punctuation=True))
print(gen_random_str(5, 7, prefix='FISHBASE_', has_digit=True, has_
↪punctuation=True))
print('---')
```

执行结果:

```
--- gen_string_by_range demo ---
q4uo6E8

FISHBASE_8uCBEUH

FISHBASE_D4wRX2.py

FISHBASE_65nqlNs

FISHBASE_3"uFm$s
---
```

fish_random.**get_random_areanote**(*zone*)

省份行政区划代码，返回下辖的随机地区名称

Param

- zone: (string) 省份行政区划代码 比如 '310000'

Returns

- random_areanote: (string) 省份下辖随机地区名称

举例如下:

```
print('--- fish_data get_random_areanote demo ---')
print(cardbin_get_bank_by_name(310000))
print('---')
```

输出结果:

```
--- fish_data get_random_areanote demo ---
徐汇区
---
```

CHAPTER 3

更多

如有好的建议，欢迎提 issue： https://github.com/chinapnr/fishbase/issues

Python Module Index

f

Index

H

I

J

L

M

P

R

S

T

Y