Скрипт пишется на языке Python версии 3.8.

## Особенности выполнения

Скрипт выполняется не из файла (без записи на диск). Из-за этого:

* технически, текст скрипта не может занимать больше 128 КиБ, иначе он не запустится;
* исключения не выдают текст строки, а лишь **её номер**.

Сам скрипт выполняется из директории ~/ips3-sandbox, и вы можете читать и записывать файлы в эту директорию. Не забудьте сохранить нужные вам файлы в конце рабочей сессии!

## Базовый скрипт

import ips # 1  
psm = ips.init() # 2  
  
# ... здесь ваш код ...  
  
psm.save\_and\_exit() # 3  
  
# команда 3 завершает скрипт,  
# после неё приказы не выполнятся

1. Импорт библиотеки API стенда.
2. Запрос на сервер и создание объекта с данными активного хода.
3. Отправка сформированных приказов на сервер и завершение скрипта. Без этой команды скрипт бесполезен.

## Получение данных

### Объекты энергосети

# обозначения типов объектов  
obj\_types = [  
 "main" # подстанции  
 "miniA", # мини-подстанции А  
 "miniB", # мини-подстанции Б  
 "solar", # солнечные электростанции  
 "wind", # ветровые электростанции  
 "houseA", # дом А  
 "houseB", # дом Б  
 "factory", # больницы  
 "hospital", # заводы  
 "storage", # накопители  
]  
  
for obj in psm.objects:  
 print("== Объект:", obj.id, "==") # (тип, номер)  
 print("Тип: ", obj.type) # см. выше  
 print("Включен:", obj.power.now.online) # bool  
 print("Тариф:", obj.contract) # float  
 print("Адрес:", obj.address) # [str]  
 print("Энергорайоны:",  
 obj.path) # [ <адрес энергорайона> ]  
 print("Доход:",  
 obj.score.now.income) # float  
 print("Расход:",  
 obj.score.now.loss) # float  
 print("Доход за первый ход:",  
 obj.score.then[0].income)  
 print("Расход за 5ый ход:",  
 obj.score.then[4].loss) # float  
 print("Генерация:",  
 obj.power.now.generated) # float  
 print("Потребление:",  
 obj.power.now.consumed) # float  
 print("Потребление за первый ход:",  
 obj.power.then[0].consumed)  
 print("Заряд (актуально для накопителя):",  
 obj.charge.now) # float  
 print("Штормовой режим (актуально для ВЭС):",  
 obj.failed) # bool

### Энергорайоны (нумерация с 1)

Энергорайоны помещены в поле networks и представляют собой словарь, где ключи — индексы (**нумеруются с 1!**), а значения — структуры, хранящие в себе информацию о соответствующих районах (состояние, показатели).

for index, net in psm.networks.items():  
 print("== Энергорайон", index, "==")  
 print("Адрес:", net.location)  
 # [ (ID подстанции, № линии) ]  
 print("Включен:", net.online) # bool  
 print("Генерация:", net.upflow) # float  
 print("Потребление:", net.downflow) # float  
 print("Потери:", net.losses) # float  
 print("Износ ветки:", net.wear) # float

### Прогнозы

На каждую величину (потребление и погода) дано по несколько прогнозов. Доступ к ним осуществляется через поле forecasts, после чего идёт обращение к типу прогнозов и индекс прогноза.

Каждый прогноз является последовательностью медиан, при это известно максимальное отклонение значения от медианы. Это отклонение общее для всех прогнозов этого типа.

# дом А, 1ый ход  
x = psm.forecasts.houseA[0] #  
# завод, 6ой ход  
x = psm.forecasts.factory[5]  
# больница, 11ый ход  
x = psm.forecasts.hospital[10]  
# солнце, 2ой ход  
x = psm.forecasts.sun[1]  
# ветер, район 1, 3ий ход  
x = psm.forecasts.wind["1"][2]  
  
# максимальное отклонение прогноза для района 2  
spr = psm.forecasts.wind["2"].spread

### Погода

print("Сила ветра в районе 3:", psm.wind["3"].now) # float  
print("Была на 1 ходу:", psm.wind["3"].then[0]) # float  
  
print("Яркость солнца:", psm.sun.now) # float  
print("Была на 5 ходу:", psm.sun.then[4]) # float

### Аварии

print("Будет ли авария на 5 ходу:",  
 psm.fails[4]) # bool

### Биржа

print("Фактические контракты:")  
for receipt in psm.exchange:  
 print("Контрагент:", receipt.source)  
 # "exchange" = оператор,  
 # "overload" = штраф за перегрузку,  
 # иначе = другой игрок  
 print("Объём:", receipt.flux)  
 # Плюс = покупка, минус = продажа  
 print("Цена за МВт:", receipt.price)  
 print("")

Игрок представлен словарём с ключами place и player.

### Прочая информация

Эти поля из объекта стенда не связаны с важными данными, но тоже могут пригодиться.

print("Ход:", psm.tick) # int  
print("Всего ходов:", psm.gameLength) # int  
print("Изменение счёта:", psm.scoreDelta) # float  
  
print("Всего сгенерировано:",  
 psm.total\_power.generated) # float  
print("Всего потреблено:",  
 psm.total\_power.consumed) # float  
print("Получено с биржи (минус = отправлено):",  
 psm.total\_power.external) # float  
print("Всего потерь:",  
 psm.total\_power.losses) # float

## Приказы

Для управления энергосистемой используются управляющие воздействия (приказы). Вы можете объявлять их с помощью функций из psm.orders. При **корректном** завершении скрипта (psm.save\_and\_exit()) эти приказы отправляются в систему.

**Приоритет приказов:**

* Линии — выполняется последний отправленный
* Заявки на биржу — выполняются все по отдельности
* Аккумулятор — выполняются все по порядку
* График — линии складываются вместе по графикам
  + Длина линии не больше числа тактов в игре
  + Не более 5 линий на график
  + 4 графика (нумеруются от 0 до 3)

**Отмена приказов невозможна!**

Все числовые параметры в приказах — ненулевые положительные!

# Отправить 10 МВт в накопитель c3  
psm.orders.charge("c3", 10)  
  
# Забрать 5 МВт из накопителя c1  
psm.orders.discharge("c1", 5)  
  
# Включить линию 2 на подстанции М2  
psm.orders.line\_on("M2", 2)  
  
# Выключить линию 1 на мини-подстанции m1  
psm.orders.line\_off("m1", 1)  
  
# Заявка на продажу 10,2 МВт за 2,5 руб./МВт  
psm.orders.sell(10.2, 2.5)  
  
# Заявка на покупку 5,5 МВт за 5,1 руб./МВт  
psm.orders.buy(5.5, 5.1)  
  
# Поместить линию из трёх точек  
# на 4-ый пользовательский график  
psm.orders.add\_graph(3, [1.2, 3.4, 5.6])

## Отладка

Для локальной проверки и отладки скрипта можно использовать локальную среду IDLE со встроенным модулем ips.

В модуле реализованы команды:

* Для замены init():
  + init\_test() — данные из вшитого примера
  + from\_json(json\_str) — данные из JSON-string
  + from\_file(filename) — данные из файла в JSON-формате
* Для замены Powerstand.save\_and\_exit:
  + print(psm.get\_user\_data()) — вывод данных из пользовательских графиков
  + print(psm.orders.get()) — вывод приказов в чистом виде в stdout без завершения скрипта
  + print(psm.orders.humanize()) — то же самое, но приказы представлены в читаемом виде

Пример отладочной версии скрипта и правок для получения оной:

import ips  
  
# psm = ips.init() # было  
psm = ips.init\_test() # стало  
  
# ... здесь ваш код ...  
  
# psm.save\_and\_exit() # было  
print("\n".join(psm.orders.humanize())) # стало  
# графики в приказах не приводятся, но можно так:  
print(psm.get\_user\_data())