# Библиотеки: nn, nlu, nv для написания логики

### NeuroNetLibrary - python библиотека, содержит общие функции

Доступна внутри звонка и диалога в виде объекта **nn** (nn = NeuroNetLibrary(nlu\_call, event\_loop))

| **Метод** | **Описание** | **Пример** |
| --- | --- | --- |
| nn.dialog | Методы для получение / изменения данных диалога  nn.dialog.result - статус диалога  nn.dialog.msisdn - номер абонента  nn.dialog.entry\_point - точка входа в логику | // завершить диалог  nn.dialog.result = nn.RESULT\_DONE  // получить номер абонента  nn.log('msisdn', nn.dialog.msisdn)  // поменять точку входа  nn.dialog.entry\_point = 'main\_2' |
| nn.env(name, val = None) | Установка и получение переменных окружений.  Сохраняется в рамках всех звонков и диалога.  Зарезервированные переменные окружения:  nn.env({  "flag": "", // Голос озвучки у промптов, используются в логики звонка  "dialog\_start\_time": "", // (readonly) время старта текущего диалога  "dialog\_uuid": "", // (readonly) uuid текущего диалога  "agent\_uuid": "", // (readonly) uuid текущего агента  "call\_start\_time": "", // (readonly, online) время запуска звонка  "call\_uuid": "", // (readonly, online) uuid звонка  }) | print(nn.env()) // print all environments  flag = nn.env('flag')  nn.env('flag', 'vova') // set flag = vova  flag = nn.env('flag') // vova  gender = nn.env('gender')  if gender == 'female':  nn.env('flag', 'vova')  // some action with male voice  else:  nn.env('flag', 'default')  lang = nn.env('lang')  if lang:  nv.media\_params('lang', lang)  // some action with ru-RU |
| nn.storage(key1, key2, ..., keyn) | Получение данных по ключам из пользовательского хранилища.  Поиск идет сначала по хранилищу агента, затем глобально по компании. Если искать только один ключ, возвращается **str**, если запрашивать несколько – **dict**. | token = nn.storage('middleware\_token')  headers = {'Authorization': 'Bearer ' + token}  response = requests.post(url, data, headers=headers, timeout=5) |
| nn.counter(name, op=None) | Получение или изменение внутреннего счетчика в логике. Используется для того, чтобы знать сколько раз был вызван тот или иной метод. Обнуляется после каждого звонка и диалога. | tail\_count = nn.counter('tail\_count') // 0  tail\_count = nn.counter('tail\_count', '+') // 1  if tail\_count == 5:  tail\_count = nn.counter('tail\_count', 1)  if tail\_count == 1:  nv.say('hello\_1') |
| nn.log() | Функция для логирования данных в статистику  nn.log(name, data) nn.log(data) | nn.log('call\_duration', nv.get\_call\_duration())  nn.log('call\_transcription', nv.get\_call\_transcription(return\_format=nv.TRANSCRIPTION\_FORMAT\_TXT)) |
| nn.call(msisdn: str,  date: (datetime, str) = None,  channel: str = None,  script: (str, UUID) = None,  entry\_point: str = None,  transport: str = 'sip',  on\_success\_call: Any[None, str] = None,  on\_failed\_call=None,  proto\_additional: dict=None) | Добавление звонка в очередь на обзвон.  Описание параметров:  msisdn: (str) номер абонента кому звонить (обязательный параметр)  date: (datetime, str) дата запланированного звонка (по-умолчанию текущая дата и время, то есть сразу)  // (str) в формате даты ('%Y-%m-%d %H:%M', '%Y-%m-%d %H:%M:%S'  // Если передать в формате ('%H:%M:%S', '%H:%M'),  // тогда звонок будет запущен через данное кол-во часов и минут  Если дата <= текущего времени, звонок будет запущен сразу.  Такое же поведение, если не передать совсем (default None)  channel: название канала (транка), через который совершается звонок.  Канал должен быть указан в CMS и доступен текущему агенту.  Если не указать, будет использоваться текущий канал или канал по-умолчанию (default None)  script: название или UUID скрипта логики, с которое начнется диалог, должен принадлежать текущему агенту.  Если не указан, используется текущий (default None)  entry\_point: названии функции (точки входа), с которой будет запущен скрипт script, (default main)  transport: голосовй транспорт, для звонков пока что используется только sip (default 'sip')  on\_success\_call: смена точки входа после успешного звонка (default None)  on\_failed\_call: смена точки входа после неудачного звонка или недозвона (default None)  proto\_additional: дополнительные хедеры, которые будут переданы в INVITE, где  ключ, значение - имя и содержимое хедера соответственно | // звонок через тот же канал, и тем же скриптом  nn.call('89001234567', '25-03-2020 01:00:00')  // звонок по SIP, через канал mtt и со скриптом second\_script  nn.call('89001234567', '25-03-2020 01:00:00', 'mtt', 'second\_script', 'sip') |

### NeuroNluLibrary - python библиотека, для обработки текста

Доступна внутри звонка и диалога в виде объекта **nlu** (nlu = NeuroNluLibrary(nlu\_call, event\_loop) )

nlu.extract - метод для выдениея сущностей и интентов, возвращает объект NeuroNluRecognitionResult (описание в nv.listen)

Пример:

r = nlu.extract('Да я хочу cashback', 'increase\_cashback')

print(r.utterance(), r.has\_entities(), r.entity('increase\_cashback'))

r = nlu\_extract('Перезвоните позже', 'callback')

print(r.utterance(), r.has\_intents(), r.intent('callback'))

result = nlu.extract(

// Текст для выделения сущностей (string)

text,

// Список сущностей для распознавания,

// если None (по-умолчанию), то поиск будет по всем сущностям текущего агента

entities=['entity1', 'entity2'],

// допускается передача string, разделенной запятыми

entities='entity1, entity2',

// Включить инверсию поиска (bool), исключаем переданные сущности

// если False (по-умолчанию), то распознование будет только по entities выше

// если True, то поиск будет по всем сущностям, кроме тех что переданы в entities

entities\_exclude=True | False),

// Список интентов для распознавания

intents=['intent1', 'intent2'], // list

intents='intent1, intent2', // или строка, разделенная запятыми

intents\_exclude=True | False), // инверсия поиска (исключить сущности)

context=None, // строка контекста для Nlu API (используется совместно с use\_neuro\_api=True)

use\_neuro\_api=False // если True, то для определения сущностей будет задействован Nlu Api.

Сущности найденные через Nlu Api не будут определяться по паттернам.

)

### NeuroVoiceLibrary - python библиотека, для написания логики во время звонка на python

Доступна внутри звонка в виде объекта **nv** nv = NeuroVoiceLibrary(nlu\_call, loop)

| **Метод** | **Описание** | **Пример** |
| --- | --- | --- |
| nv.say(name, val = None) | Проигрывание аудио-файла промпта по названию name, или сущности с названием name или 'default' со значением val, в обоих случаях файл берется из БД по параметрам: lang, flag и agent\_id |  |
| nv.has\_record(name, val = None) | Проверяет существование аудио файла для проигрывания по текущему nn.env('flag'), параметры одинаковые как в nv.say() |  |
| nv.media\_params(name, val = None) | Установка и получение параметров медиа сервера. Доступные параметры медиа сервера:  nv.media\_params({  "asr": "", // Движок распознования (google yandex tinkoff)  "tts": "", // Движок синтеза (google yandex tinkoff)  "lang": "", // Язык для распознования, синтеза и проигрывания промптов (ru-RU, en-US и тд)  "authentication\_data": {"asr": {...}, "tts": {...}} // Токены в сервисах распознования и синтеза  }) | nv.media\_params('lang', 'ru\_RU')  lang = nv.media\_params('lang') // ru\_RU  nv.media\_params({'asr': 'google', 'tts': 'yandex'})  current\_asr = nv.media\_params('asr') // google  current\_tts = nv.media\_params('tts') // yandex |
| nv.listen() | Запуск распознавания и выделения сущностей | Пример:  with nv.listen(  // Правила детектирования ввода и остановки (tuple, string, integer или None),  // используется для отмены текущего воспроизведения аудио.  // Eсли передано оба аргумента, остановка сработает только в случае если оба правила сработали  // Eсли не указывать (по-умолчанию), то остановки проигрывания не будет  detect\_policy = ( // touple  stop\_entities, // (string, list) опциональный, остановка по сущностям, разделенные ",", должны быть в списке entities ниже  stop\_intents, // (string, list) опциональный, остановка по интентам, разделенные ",", должны быть в списке intents ниже  stop\_characters\_count, // (integer) опциональный, кол-во распознанных символов  stop\_condition, // (string) опциональный, условие OR или AND (по-умолчанию AND)  ),  // допускается короткий синтаксис  detect\_policy = 'entity1, entity2', // (string разделенная запятыми) только stop\_entities  detect\_policy = ['entity1', 'entity2'], // (list) только stop\_entities  detect\_policy = 500, // (integer) только stop\_characters\_count  // можно передавать tuple первым аргументом без detect\_policy=  ('entity1, entity2', 'intent1, intent2', 500, 'OR'),  // Список сущностей для распознавания,  // если None (по-умолчанию), то поиск будет по всем сущностям текущего агента  entities=['entity1', 'entity2']  // допускается передача string, разделенной запятыми  entities='entity1, entity2'  // Инверсированный поиск сущностей, поиск по всем сущностям, кроме тех, что переданы  // Этот параметр имеет бОльший приоритет, чем intents  entities\_exclude='intent1,intent2'  // Список интентов для распознавания,  // если None (по-умолчанию), то поиск будет по всем сущностям текущего агента  intents=['intent1', 'intent2']  // допускается передача string, разделенной запятыми  intents='intent1,intent2'  // Инверсированный поиск интентов, поиск по всем интентам, кроме тех, что переданы  // Этот параметр имеет бОльший приоритет, чем intents  intents\_exclude='intent1,intent2'  // строка контекста для Nlu API (используется совместно с use\_neuro\_api=True)  context=None,  // если True, то для определения сущностей будет задействован Nlu Api.  // Сущности найденные через Nlu Api не будут определяться по паттернам.  use\_neuro\_api=False  // свои таймауты для распознавания текущей сессии,  // если не указать будут использованы по-умолчанию, те что в nv.set\_default('detect'),  // задавать можно в любом порядке  no\_input\_timeout=4000,  recognition\_timeout=30000,  speech\_complete\_timeout=1500,  asr\_complete\_timeout=2500  ) as r:  // some actions with r: NeuroNluRecognitionResult  // some methods after detect\_speech\_stop with result |
| nv.speech\_input\_detector=my\_input\_detector | Переопределить функцию остановки ввода на свою |  |
| nv.background(name) | Запуск проигрывания записи **name** в фоне, будет зациклена |  |
| nv.background(None) | Остановка проигрывания фоновой записи |  |
| nv.synthesize(text, ssml: True|False) | Проигрывание синтеза, флаг ssml (по-умолчанию False) включает синтаксис ssml, параметры синтеза беруться из media\_params |  |
| nv.random\_sound(min\_delay: int, max\_delay: int) | Включение проигрывания рандомных звуков, от min\_delay до max\_delay. Работает только внутри nv.listen() | with nv.listen() as r:  nv.random\_sound(2000, 7000)  nv.say('hello')  // тоже самое, но используя параметры по-умолчанию  nv.set\_default('random\_sound', min\_delay=2000, max\_delay=7000)  with nv.listen() as r:  nv.random\_sound()  nv.say('hello') |
| nv.get\_call\_duration()-> int | Получение текущей длительности звонка | nn.log('call\_duration', nv.get\_call\_duration()) |
| nv.get\_call\_transcription(return\_format) → dict, str | Получение текущей транскрипции звонка | nv.get\_call\_transcription(return\_format=nv.TRANSCRIPTION\_FORMAT\_TXT) → str  nv.get\_call\_transcription() → dict  nn.log('call\_transcription', nv.get\_call\_transcription(return\_format=nv.TRANSCRIPTION\_FORMAT\_TXT)) |

### Result – ссылка на объект NeuroNluRecognitionResult

| **Методы** | **Описание** |
| --- | --- |
| result.utterance() | Распознанный текст, очищенный от левых символов и сущностей preprocess\_expressions |
| result.entity('entity\_name') | Возвращает значение сущности, если сущности не существует вернет None |
| result.has\_entity('entity\_name') | Наличие сущности (True, False) |
| result.has\_entities() | Наличие любых найденных сущностей (True, False) |
| result.intent('intent\_name') | Возвращает значение интента, если интента не существует вернет None |
| result.has\_intent('intent\_name') | Наличие интента (True, False) |
| result.has\_intents() | Наличие любых найденных интентов (True, False) |

Примеры:

# правило остановки только по кол-ву символов

with nv.listen(

(None, 500) # 0 аргумент tuple(stop\_entities, stop\_character\_count)

) as r:

# тоже самое но без tuple, передаем integer без скобок

with nv.listen(

500 # 0 аргумент integer – stop\_character\_count

) as r:

# правило остановки только по сущностям

with nv.listen(

'entity1,entity2' # 0 аргумент string – stop\_entities

) as r:

# правило остановки только по сущностям + распознование только определенных

with nv.listen(

'confirm,callback', # 0 аргумент (string разделенная запятыми) – stop\_entities,

entities='hello\_confirm,confirm,confirm' # именованный аргумент entities, string разделенная запятыми – сущности для распознавания

) as r:

# без правил остановки, передаем свой таймаут и сущности для распознования

# порядок именованных аргументов не важен

with nv.listen(

recognition\_timeout=60000, # меняем recognition\_timeout на 60 сек, остальные параметры берутся из set\_default

entities='hello\_confirm,confirm', # именованный аргумент entities string разделенная запятыми – сущности для распознавания

recognition\_timeout=60000 # меняем recognition\_timeout на 60 сек, остальные параметры берутся из set\_default

) as r:

# без правил остановки, исключаем из распознавания сущности voice\_mail

with nv.listen(

entities='voice\_mail', # именованный аргумент entities string разделенная запятыми – сущности для распознавания

entities\_exclude=True # исключаем переданные сущности, то есть распознаем все, кроме voice\_mail

) as r: