Домашнее задание №7

Дедлайн 1 (16 баллов): 25 октября, 13:00
Дедлайн 2 (8 баллов): 1 ноября, 13:00

Домашнее задание нужно написать на Python 3 и сдать в виде одного файла. Правило именования файла: `name_surname_07.py`. Например, если вас зовут Иван Петров, то имя файла должно быть: `ivan_petrov_07.py`.

Перед отправкой решения убедитесь, что оно соответствует рекомендациями из списка pydonts: `https://github.com/superbobry/pydonts`.

1 В следующих заданиях нельзя использовать классы и функции из модуля `contextlib`.

а Реализуйте менеджер контекста `assert_raises`, который проверяет, что в результате выполнения тела оператора `with` поднимается исключение указанного класса.

- Если исключение не было поднято, менеджер поднимает `AssertionError`.
- Если было поднято исключение другого класса, оно пропускается менеджером.

Пример использования менеджера `assert_raises`:

```
>>> with assert_raises(ValueError):
...     "foobar".split("")
...
>>> with assert_raises(ValueError):
...     pass
...
AssertionError: did not raise 'ValueError'
>>> with assert_raises(ValueError):
...     raise TypeError
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
TypeError
```

б Реализуйте менеджер контекста `closing`, принимающий экземпляр класса, в котором определён метод `close`. При выходе из контекста менеджер должен вызывать этот метод у экземпляра.

```
>>> with closing(open("example.txt")) as handle:
...     copy = handle
...
>>> copy.closed
True
```

в Реализуйте менеджер контекста `log_exceptions`, который выводит исключения, возникшие при выполнении тела оператора `with` в `sys.stderr`:

```
>>> def f():
...     with log_exceptions():
...         {}["foobar"]
...     return 42
...
>>> f()
Traceback (most recent call last):
```

```
  File "<stdin>", line 3, in f
KeyError: 'foobar'
42
```

Обратите внимание, что менеджер подавляет возникшие исключения, поэтому функция `f` возвращает значение.

Для вывода на экран информации об исключении нужно использовать функцию `print_exception` из модуля `traceback`:

```
>>> import traceback
>>> try:
...     {}["foobar"]
... except KeyError as exc_value:
...     exc_type = exc_value.__class__
...     tb = exc_value.__traceback__
...     traceback.print_exception(exc_type, exc_value, tb)
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyError: 'foobar'
```

г Реализуйте декоратор `with_context`, который принимает один аргумент — менеджер контекста и возвращает новую функцию, которая вызывает исходную в указанном контексте. Пример:

```
>>> from contextlib import redirect_stdout
>>> import io
>>> handle = io.StringIO()
>>> @with_context(redirect_stdout(handle))
... def f():
...     print("Hello world!")
...
>>> f()
>>> handle.getvalue()
'Hello world!\n'
```

Ook!

Язык Ook![1] был спроектирован австралийским физиком Девидом Морганом-Маром специально для удобства орангутанов: в языке всего три базовых синтаксических элемента, каждый из которых понятен любому орангутану:

- `Ook.`
- `Ook?`
- `Ook!`

Программа на языке Ook! работает с массивом целых чисел фиксированного размера. Перед работой программы все элементы массива обнуляются. `MP` (memory pointer) указывает на первую ячейку массива. Нумерация ячеек начинается с нуля.

В языке восемь возможных инструкций:

- `Ook. Ook?` Передвинуть `MP` в следующую ячейку массива.
- `Ook? Ook.` Передвинуть `MP` в предыдущую ячейку массива.
- `Ook. Ook.` Увеличить значение в ячейке массива, на которую указывает `MP`, на единицу.
- `Ook! Ook!` Уменьшить значение в ячейке массива, на которую указывает `MP`, на единицу.
- `Ook! Ook.` Напечатать символ с ASCII кодом, равным значению ячейки, на которую указывает `MP`.
- `Ook. Ook!` Прочитать символ со стандартного входа (`sys.stdin.read(1)`) и записать его ASCII код в ячейку, на которую указывает `MP`.
- `Ook! Ook?` Перейти к инструкции, следующей за `Ook? Ook!`, если значение, на которое указывает `MP` равняется нулю, иначе перейти к следующей по порядку инструкции.
- `Ook? Ook!` Перейти к инструкции, следующей за `Ook! Ook?`, если значение, на которое указывает `MP` не равняется нулю, иначе перейти к следующей по порядку инструкции.

Об инструкциях `Ook! Ook?` и `Ook? Ook!` удобно думать в терминах цикла **`while`** из языка C (открывающая и закрывающая скобка соответственно, то есть место проверки условия цикла):

```
while (memory[ptr]) {  // Ook! Ook?
    // ...
}                      // Ook? Ook!
```

Обратите внимание, что циклы **могут быть** вложенными. При этом команды `Ook! Ook?` и `Ook? Ook!` должны образовывать правильную скобочную последовательность.

Пример короткой программы на языке Ook!:

```
Ook. Ook! Ook! Ook.
```

Программа читает ASCII символ со стандартного ввода, а затем печатает его.

[1] `http://www.dangermouse.net/esoteric/ook.html`

В силу своей простоты язык Ook! стал довольно популярен, поэтому мы решили реализовать распределенный интерпретатор языка Ook!. Ваша задача — написать бэкенд на Python, который будет работать на всех узлах распределенной системы.

Бэкенд можно условно разделить на две части: интерпретатор и коммуникатор.

2 Задача интерпретатора — выполнить исходный код на языке Ook!.

а Объявите класс `Op` — перечисление возможных инструкций языка Ook!. Каждая инструкция представляется парой строк. Имена инструкций и общий вид класса приведён в примере ниже.

Для объявления класса-перечисления в Python используется базовый класс `Enum` из модуля `enum`[2].

```
>>> from enum import Enum
>>> class Op(Enum):
...     NEXT = ("Ook.", "Ook?")
...     PREV = ("Ook?", "Ook.")
...     INC = ...
...     DEC = ...
...     PRINT = ...
...     INPUT = ...
...     START_LOOP = ("Ook!", "Ook?")
...     END_LOOP = ("Ook?", "Ook!")
...
```

Краткий ликбез по перечислениям в Python:

- к элементам перечисления можно обращаться как к атрибутам класса,

  ```
  >>> Op.NEXT
  <Op.NEXT: ('Ook.', 'Ook?')>
  ```

- связанное с элементом перечисления значение доступно через атрибут `value`,

  ```
  >>> Op.NEXT.value
  ('Ook.', 'Ook?')
  ```

- для сравнения двух элементов перечисления нужно использовать оператор `is`,

  ```
  >>> Op.NEXT is Op.NEXT
  True
  ```

- получить все элементы перечисления можно, проитерировавшись по классу.

  ```
  >>> {op.value: op for op in Op}
  {('Ook.', 'Ook?'): <Op.NEXT: ('Ook.', 'Ook?')>, ...}
  ```

б Реализуйте функцию `ook_tokenize`. Функция принимает на вход строчку — исходный код на языке Ook! и возвращает последовательность инструкций, элементов перечисления `Op`.

```
>>> ook_tokenize("Ook. Ook! Ook! Ook.")
[<Op.INPUT: ('Ook.', 'Ook!')>, <Op.PRINT: ('Ook!', 'Ook.')>]
```

Можно считать, что исходный код состоит из инструкций языка Ook!, разделенных одним пробелом.

[2] `https://docs.python.org/3/library/enum`

в Реализуйте функцию `ook_eval`, интерпретирующий исходный код на языке Ook!. Размер массива, с которым работает программа на Ook!, должен быть только-ключевым аргументом со значением по умолчанию `2**16`.

Можно считать, что входная программа корректна и всегда завершается, то есть в ней отсутствуют: ошибки печати и чтения ASCII символов, ошибки выхода за границы массива, зацикливания.

Общий вид функции `ook_eval`:

```
def ook_eval(source, *, memory_limit=2**16):
        code = ook_tokenize(source)
        memory = [0] * memory_limit
        mp = 0  # Memory Pointer.
        # ...
```

Несколько примеров использования интерпретатора:

```
>>> ook_eval("Ook. Ook! Ook! Ook.")
!  # читается с sys.stdin.
!
>>> ook_eval("\
... Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. \
... Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. \
... Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook? Ook! Ook! Ook? Ook! Ook? Ook. \
... Ook! Ook. Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. \
... Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook? \
... Ook! Ook! Ook? Ook! Ook? Ook. Ook. Ook. Ook! Ook. Ook. Ook. Ook. Ook. Ook. Ook. \
... Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook. Ook! Ook. Ook. Ook. Ook. Ook. \
... Ook. Ook. Ook! Ook. Ook. Ook? Ook. Ook? Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. \
... Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. \
... Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook? Ook! Ook! Ook? Ook! Ook? Ook. Ook! Ook. \
... Ook. Ook? Ook. Ook? Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. \
... Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. \
... Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. \
... Ook. Ook? Ook! Ook! Ook? Ook! Ook? Ook. Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook. \
... Ook? Ook. Ook? Ook. Ook? Ook. Ook? Ook. Ook! Ook. Ook. Ook. Ook. Ook. Ook. Ook. \
... Ook! Ook. Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook. \
... Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! \
... Ook! Ook. Ook. Ook? Ook. Ook? Ook. Ook. Ook! Ook.")
Hello World!  # https://gist.github.com/superbobry/f9dd3a66afe03e3935d0
```

Пример более сложной программы на языке Ook! можно найти по ссылке: `https://gist.github.com/superbobry/9f95d91eba71932411ae`. Программа выводит на экран фразу из 128-го сонета Вильяма Шекспира: Making dead wood more bless'd than living lips.

3 Коммуникатор отвечает за общение с сервером заданий нашей распределенной системы. Общий вид класса `Pipe`, реализующего функциональность коммуникатора:

```
class Pipe:
    def __init__(self, server_url):
        self.server_url = server_url

    def pull(self):
        pass

    def push(self, token, result):
        pass

    def loop(self, n_iter, interval=15):
        pass
```

У конструктора `Pipe` один параметр: `server_url` — URL сервера заданий.

а Реализуйте метод `pull`, который получает очередное задание от сервера. Метод должен делать GET запрос по адресу `server_url`. Ответ состоит из двух строк:

```
AZDAFX
Ook. Ook! Ook! Ook.
```

Первая строка ответа — уникальный идентификатор задания, состоящий ровно из шести букв латинского алфавита в верхнем регистре. Вторая строка — исходный код программы на Ook!, который нужно интерпретировать.

Метод `pull` должен вернуть пару **строк**: токен и исходный кода на языке Ook!.

```
>>> Pipe("http://ook.compscicenter.ru").pull()
('AZDAFX', 'Ook. Ook! Ook! Ook.')  # У вас может получиться другой результат.
```

Модуль `urllib.request` позволяет общаться с серверами по протоколу HTTP. Выполнить GET запрос можно с помощью функции `urlopen`[3]:

```
>>> from urllib.request import urlopen
>>> with urlopen("http://ya.ru") as page:
...     print(page.read()[:32])
...
b'<!DOCTYPE html><html lang="ru"><'
```

Объект, возвращаемый функцией `urlopen` реализует интерфейс файла, открытого в режиме чтения байтов.

Обратите внимание, что реализованный нами интерпретатор не работает с байтами, поэтому вам нужно будет воспользоваться методом `bytes.decode`. Можно считать, что HTTP сервер использует кодировку ASCII.

б Реализуйте метод `push`, который отправляет результат вычисления программы на языке Ook! обратно на сервер заданий. Метод принимает два аргумента: идентификатор задания, полученный от сервера, и результат вычисления программы. Оба аргумента — строки. Метод должен делать POST запрос по адресу `server_url` с телом, состоящим не менее чем из двух строк:

```
TOKEN
RESULT
```

Выполнить POST запрос с помощью `urlopen` можно следующим образом:

[3] https://docs.python.org/3/library/urllib.request

```
>>> data = b""  # Тело POST запроса.
>>> with urlopen("http://ya.ru", data) as page:
...     print(page)
...
Traceback (most recent call last):
urllib.error.HTTPError: HTTP Error 403: Forbidden
```

В данном случае ошибка — это нормально: сервер Яндекса не готов принимать от нас POST запросы. Обратите внимание, что второй аргумент функции `urlopen` должен быть типа `bytes`.

в Реализуйте метод `loop`, который принимает два аргумента: количество заданий, которое нужно выполнить, и интервал (в секундах), с которым коммуникатор будет спрашивать задания у сервера.

Логика работы метода следующая:

1. запомним текущее время `start_time`,
2. получим новое задание от сервера,
3. выполним его с помощью интепретатора `Ook`, перехватив стандартный поток вывода,
4. отправим на сервер напечатанную программой строку,
5. запомним текущее время `end_time`,
6. если мы ещё не выполнили нужное количество заданий, то подождём не менее, чем `max(0, interval - (end_time - start_time))` и перейдём к пункту 1.

Для перехвата стандартного потока вывода удобно воспользоваться менеджером контекста `redirect_stdout` из модуля `contextlib`. Пример его использования из слайдов лекции:

```
>>> from contextlib import redirect_stdout
>>> import io
>>> handle = io.StringIO()
>>> with redirect_stdout(handle):
...     print("Hello, World!")
...
>>> handle.getvalue()
'Hello, World!\n'
```

Также вам могут быть полезны функции `perf_counter` и `sleep` из модуля `time`[4].

- Функция `perf_counter` возвращает время в секундах от фиксированой точки в прошлом.

  ```
  >>> time.perf_counter()
  343789.46225825005
  ```

- Функция `sleep` делает ровно то, что обещает название: приостанавливает выполнение программы на указанное количество секунд.

Проверить работоспособность коммуникатора можно, соединив его с сервером заданий по адресу `http://ook.compscicenter.ru`:

```
>>> Pipe("http://ook.compscicenter.ru").loop(5)
```

Если хотите, вы можете принять участие в сборе статистики, включив свою фамилию в URL сервера заданий, например:

```
>>> Pipe("http://ook.compscicenter.ru/petrov").loop(5)
```

[4] https://docs.python.org/3/library/time

4 Коммуникатор уже почти готов к использованию. Единственное, что нам осталось — сделать его более устойчивым к ошибкам.

а Вынесите работу с функцией `urlopen` в метод `_request`:

```
class Pipe:
    # ...
    def _request(self, data=None):
        pass
```

Для GET запроса аргумент `data` должен иметь значение `None`, а для POST запроса `data` — это тело POST запроса (последовательность байт из не менее, чем двух строк).

Измените методы `Pipe.pull` и `Pipe.push` таким образом, чтобы они использовали метод `Pipe._request`, а не функцию `urlopen`. Убедитесь, что метод `Pipe._request` использует оператор `with` для безопасной работы с ответами сервера заданий.

б В соответствии с протоколом HTTP[5] у ответа HTTP сервера есть численный код, называемый статусом.

- Статусы 2XX означают, что запрос был принят и обработан сервером.
- Статусы 4XX — что запрос был составлен некорректно, например, для POST запросов это может означать отсутствие обязательных параметров.
- Статусы 5XX — что сервер не смог обработать корректный запрос.

Для статусов, отличных от 2XX функция `urlopen` поднимает исключение типа `HTTPError`. Статус ответа доступен в атрибуте `code`:

```
>>> from urllib.error improt HTTPError
>>> try:
...     with urlopen("http://ya.ru", b"") as page:
...         pass
... except HTTPError as e:
...     print(e.code)
...
403
```

Измените код метода `Pipe._request` чтобы он обрабатывал исключение `HTTPError`. Если статус ответа 4XX, то исключение нужно переподнять, если 5XX — повторить запрос снова. В случае другого статуса нужно поднять исключение типа `RuntimeError`:

```
raise RuntimeError("unexpected status code") from e
```

Список возможных ответов сервера:

- Статус 403 возвращается в том случае, если сервер ничего не знает про токен. Сервер забывает про токен **спустя десять секунд** после его выдачи.
- В том случае, если формат запроса оказывается неожиданным для сервера, возвращается статус 400 с соответствующим пояснением в теле ответа.
- Также статус 400 с текстом `False` в теле ответа возвращается, если сервер не согласен с ответом.
- Бывает, что на сервере что-то выходит из строя, тогда возвращается статус 500.

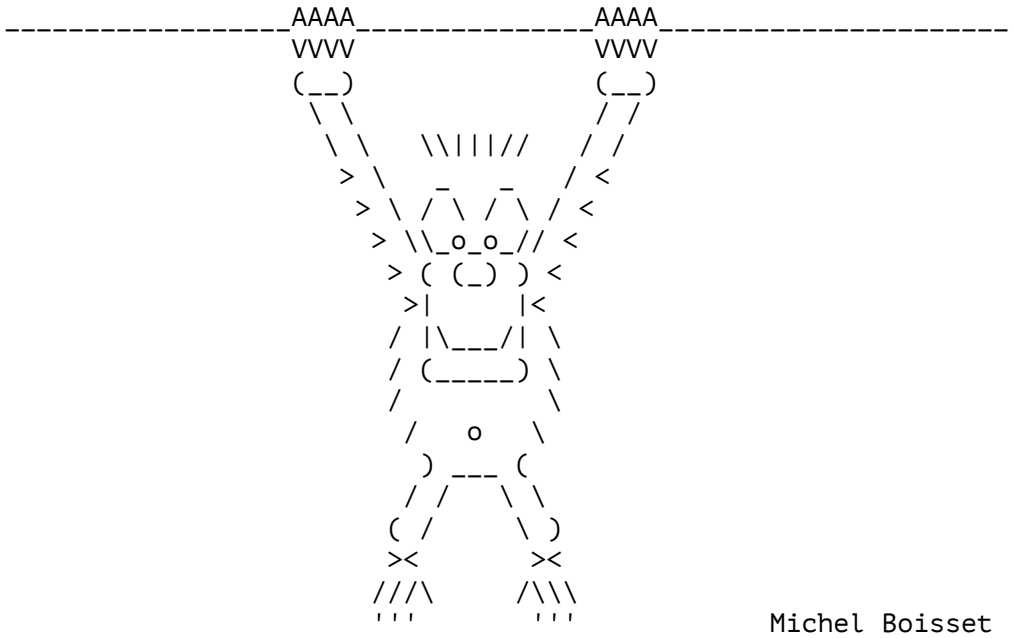
[5] `http://en.wikipedia.org/wiki/List_of_HTTP_status_codes`

в Кроме ошибок уровня протокола могут случиться и более прозаичные вещи, например, сервер может потерять доступ к интернету и соединение будет закрыто по таймауту. Функция `urlopen` поднимает такие ошибки в виде экземпляров `URLError`.

Добавьте в метод `Pipe._request` обработку исключений `URLError`. При возникновении такой ошибки метод должен повторить запрос ещё раз.

Исключение `URLError` нужно импортировать из модуля `urllib.errors`.

г Поздравляю, бэкенд для распределенного интерпретатора языка Ook! готов!



Дополнительные задания

Задачи со звёздочкой не оцениваются дополнительными баллами. Вы можете реализовать исключительно для своего удовольствия.

5* Предположение о корректности программы на Ook!, которое мы сделали при реализации интепретатора языка, не очень реалистично. Попробуем исправить его с помощью исключений.

а Объявите класс `OokException` — базовый класс для исключений интерпретатора и унаследуйте от него следующие исключения:

- `OokSyntaxError` — исходный код, переданный методу `ook_eval` не является корректным с точки зрения синтаксиса языка Ook!,
- `OokIndexError` — указатель `MP` вышел за границы массива,
- `OokValueError` — интерпретатор попытался напечатать или прочитать из `sys.stdin` значение, не являющееся кодом ASCII символа.

б Добавьте проверки на корректность в методы `ook_tokenize` и `ook_eval`. Используйте подходящие по смыслу исключения.

Примеры некорректных программ на языке Ook!:

```
# Ошибка синтаксиса.
Ook? Ook? Ook? Ook!

# Выход за границы массива.
Ook? Ook. Ook? Ook.

# Вывод на экран ASCII символа с отрицательным кодом.
Ook! Ook! Ook! Ook.

# Незакрытый цикл, то есть фактически ошибка синтаксиса.
Ook! Ook? Ook. Ook?
```

6* Реализуйте функцию `ook_encode`, которая принимает на вход строку `s` из символов ASCII и возращает исходный код программы на языке Ook!. При вычислении полученного исходного кода на экран должна выводиться строка `s`.