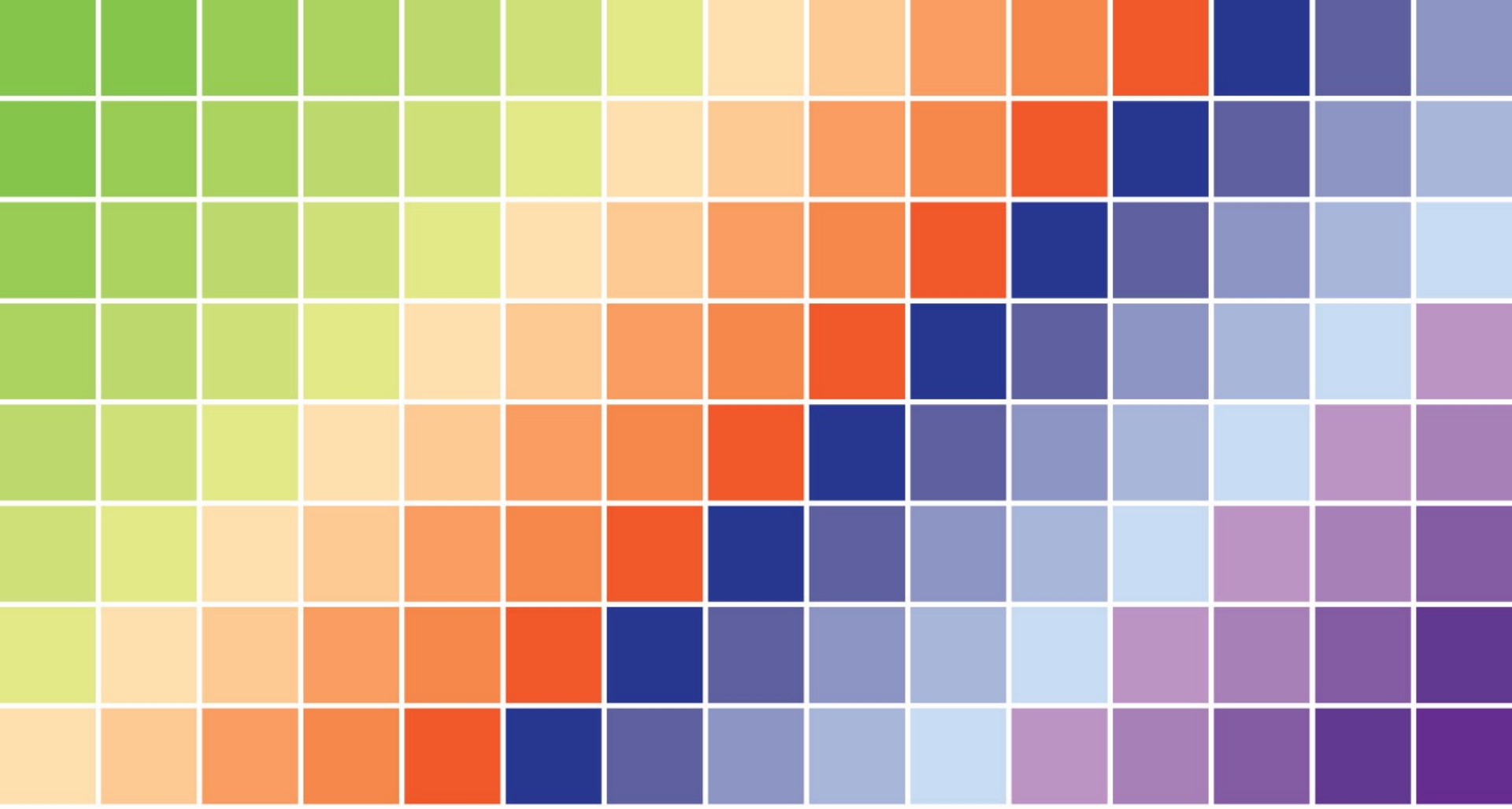


School of Computer Science

NONE. IS И IS NOT. СПИСКИ. ЦИКЛ FOR

ПРОГРАММИРОВАНИЕ НА PYTHON

Лекции для IT-школы

ВОПРОС ПО ПРОШЛОМУ ЗАНЯТИЮ

Схема оператора цикла:

```
while <логическое выражение>:
    Код, выполняемый при True
    для логического
    выражения – тело цикла
else:
    Блок кода, выполняемый при
    некоторых условиях
```

При каких условиях выполнится блок кода после else: в этом операторе?

ВОПРОС ПО ПРОШЛОМУ ЗАНЯТИЮ

Рассмотрите этот код:

```
count = 0
# Начало цикла
while True:
    count += 1
    if count == 5:
        continue
    elif count > 10:
        break
else:
    # Блок else цикла
    count = 28
# Конец цикла
print(count)
```

Куда передает управление:
- **continue**
- **break**

?

Что напечатает эта программа ?

ПРОВЕРОЧНЫЙ ВОПРОС

Сколько всего знаков * будет выведено после исполнения фрагмента программы?

```
1  i = 0
2  while i < 5:
3      print('*')
4      if i % 2 == 0:
5          print('**')
6      if i > 2:
7          print('***')
8      i = i + 1
9
```

ЗНАЧЕНИЕ NONE. ОСОБЕННОСТИ

- None – значение типа NoneType
  - Можно сравнивать на None с помощью операций is и ==
  - Сравнение с помощью is предпочтительней. Почему так, можно прочитать здесь: https://pythonworld.ru/tipy-dannyx-v-python/none.html
  - NoneType не поддерживает другие операции: +, -, *, >, < ,…
- Функции (методы), не возвращающие значения, в действительности, возвращают значение None

ЗНАЧЕНИЕ NONE. ПРИМЕРЫ

```
>>> none_var = None   # начальное объявление переменной, которая
>>>                   # еще не получила определенное значение

>>> type(none_var)    # узнаем тип none_var
<class 'NoneType'>
>>>
>>> none_var == None # проверим, что переменная имеет значение None
True
>>> none_var is None # еще раз проверим то же самое более надежным способом
True
>>> impossible_var = none_var + 5
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    impossible_var = none_var + 5
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
>>>
>>> lst = [1, 2, 3]
>>> empty_var = lst.append(4)  # метод list.append() возвращает None
>>> empty_var is None
True
>>> lst
[1, 2, 3, 4]
```

ОПЕРАТОРЫ IS И IS NOT

Проверяют ссылаются ли 2 объекта на одно и то же место в памяти:

```
>>> lst1 = [2, 4, 6, 8]      # список чисел
>>>
>>> lst1 = [2, 4, 6, 8]      # lst1 - ссылка на список чисел
>>> lst2 = lst1               # lst2 - ссылка на тот же самый список
>>> lst2 is lst1              # проверим это
True
>>>
>>> var1 = 3
>>> var2 = var1               # var2 и var1 ссылаются на одну область памяти
>>> var2 is var1              # проверим это
True
>>> var1 = 5                  # теперь var1 ссылается на другую ячейку памяти
>>> var2 is var1              # проверим это
False
>>>
>>> lst3 = lst1.copy()        # lst3 - ссылка на новый список с теми же значениями
>>> lst3 is not lst1          # проверим это
True
```

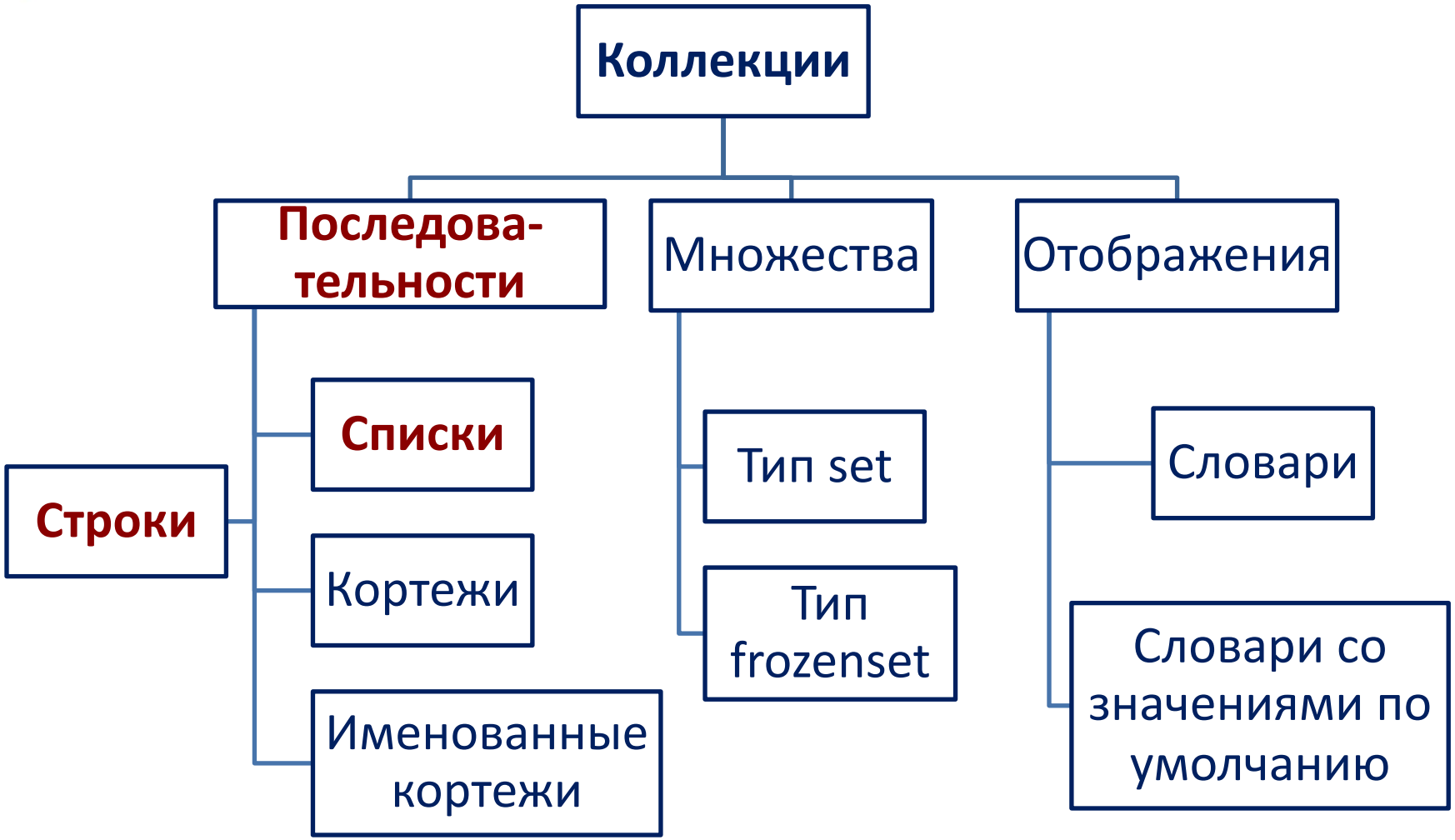
ПРИМЕР ИСПОЛЬЗОВАНИЯ NONE

- Последовательное чтение параметра в программе «Угадай число»:

```
>>> import sys
>>>
>>> max_tries = None # максимальное число попыток пока неизвестно
>>>
>>> if len(sys.argv) > 1:  # был указан параметр командной строки
        # считать max_tries из sys.argv[]
        ...

>>> if max_tries is None:  # число попыток все еще не определено
        # считать max_tries с клавиатуры
        ...
>>> ...
```

ТИПЫ КОЛЛЕКЦИЙ В PYTHON



ПОСЛЕДОВАТЕЛЬНОСТИ

- Составной тип данных, поддерживающий следующие операции:
  - Конкатенация и тиражирование: + и *
  - Проверки на вхождение in и not in
  - Функция определения размера len(object)
  - Индексация: object[index]
  - Извлечение срезов: object[start:stop:step]
  - Итерации, гарантирующие строгую последовательность элементов

- Примеры последовательностей:
  - str, list, tuple

СПИСКИ

- Объявление списка:
  - list()
  - []
  - [*элементы ч/з запятую*]

- Примеры:

```
- lst1 = list()   # пустой список
- lst2 = []       # тоже пустой список
- lst3 = [-17.5, 'text', 83]
- lst4 = [['Monday', -18], # список
          ['Tuesday', +5], # с вложен-
           'Unknown' ] # ными списками
```

МЕТОДЫ СПИСКОВ

Вызов	Описание
L.append(x)	Добавляет элемент x в конец списка L. Возвращает None
L.count(x)	Возвращает число вхождений элемента x в список L
L.extend(m) **L += m**	Добавляет в конец списка L все элементы итерабельного объекта m; то же что и ‘+=’. Возвращает None
L.index(x [, start, end])	Возвращает индекс самого первого (слева) вхождения элемента x в список L (или в срез start:end списка L) или возбуждает исключение ValueError

МЕТОДЫ СПИСКОВ

Вызов	Описание
L.insert(i,x)	Вставляет элемент x в список L в позицию int i. Возвращает None
L.pop()	Удаляет самый последний элемент из списка L и возвращает его же
L.pop(i) **del L[i]**	Удаляет из списка L элемент с индексом int i и возвращает его же. del L[i] удаляет i-ый элемент, не возвращая его
L.remove(x)	Удаляет первый найденный элемент x из списка L или возбуждает исключение ValueError, если элемент x не будет найден. Возвращает None

МЕТОДЫ СПИСКОВ

Вызов	Описание
L.clear()	Полностью очищает список. Возвращает None
L.copy()	Возвращает поверхностную копию списка
L.reverse()	Переставляет в памяти элементы списка в обратном порядке
L.sort()	Сортирует список в памяти. По умолчанию, в порядке возрастания элементов. Возвращает None. Элементы списка должны быть однотипными, иначе – TypeError

СПИСКИ И СТРОКИ

- Преобразование из строки в список:
  - **str.split(sep=None, maxsplit=-1) → list**
  - Возвращает список слов в строке, используя sep как разделитель слов
  - Пример: '1,2,3'.split(',') → ['1', '2', '3']
- Слияние списка в строку:
  - **str.join(the_string_list) → str**
  - Возвращает строку, в которую сливаются все элементы списка через заданный разделитель
  - Пример: '.'.join(['ab', 'pq', 'rs']) → 'ab.pq.rs'
- Смотрите примеры по ссылке http://pythontutor.ru/lessons/lists/#section_2 в главе «2. Методы split и join»

ПРАКТИЧЕСКОЕ ЗАДАНИЕ №1. СПИСКИ

- Пример работы со списком смотрите в shop_list.py
- Пример генерации пароля через случайный выбор символов в списке смотрите в awful_password.py
- **Дорешайте задачу со списком целых чисел в скрипте average_template.py**

ИЗМЕНЯЕМЫЕ И НЕИЗМЕНЯЕМЫЕ ПОСЛЕДОВАТЕЛЬНОСТИ

- Строки – **неизменяемые** (immutable):
  - Нельзя присваивать элемент строки
    string[n] = ‘X’
  - Индексация допустима только справа от оператора присваивания

- Списки – **изменяемые** (mutable):
  - Можно присваивать (изменять) существующий элемент списка

- Рассмотрим примеры

ЦИКЛ FOR

```
for <переменная> in
    <итерабельный объект>:
    код, выполняемый для
    каждого элемента
    итерабельного объекта…
[else:
    код, выполняемый, если блок
    for нормально завершился
    или не выполнялся вовсе
]
```

Отступы обяза-тельны!

Допустимы **break** и **continue** в теле цикла, а также **else** в конце. Смысл тот же, что и для **while**

ЦИКЛ FOR. ИТЕРАЦИИ

```
>>>
>>> # итерации по строке, т.е. перебор всех символов строки
>>> word = "Python"
>>> for letter in word:
        print(letter)


P
y
t
h
o
n
>>>
>>> # итерации по списку, т.е. перебор всех символов строки
>>> day_schedule = ["Wake-up", "Coffee", "Work", "Lunch", "Work", "Home"]
>>> for item in day_schedule:
        print(item)


Wake-up
Coffee
Work
Lunch
Work
Home
>>>
```

RANGE. СПОСОБ ГЕНЕРАЦИИ ИНДЕКСОВ ДЛЯ ПОСЛЕДОВАТЕЛЬНОСТИ

- range() генерирует индексы, с помощью которых можно адресоваться к элементам последовательности

```
range([start,] stop [, step])
→ <итерируемый объект>
```

Возвращает последовательность чисел от start (включая) до stop (исключая) с шагом step

ИСПОЛЬЗОВАНИЕ RANGE ДЛЯ ЦИКЛА FOR

- range() генерирует индексы, с помощью которых можно адресоваться к элементам последовательности:

```
for index in range([start,] stop
[, step]):
    <тело цикла, использующее
     переменную index>
```

range() *возвращает последовательность чисел от* ***start*** *до* ***stop*** *с шагом* ***step***

RANGE. ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ

```
range([start,] stop [, step])
```

– Возвращает целочисленный итератор:

1. С одним аргументом (stop) итератор представляет последовательность целых чисел от 0 до (stop-1)
2. С двумя аргументами (start, stop) – последовательность целых чисел от start до (stop-1) с шагом 1
3. С тремя аргументами – цепочку целых чисел от start до (stop-1) с шагом step

RANGE() ПОХОЖ НА СПИСОК

- range() возвращает последовательность чисел от start до stop с шагом step
- Эта последовательность легко превращается в список:

```
>>> lst = [2, 4, 6, 8]
>>> rng = range(2, 10, 2)
>>> rng
range(2, 10, 2)
>>> lst == rng
False
>>> lst == list(rng)
True
>>> list(rng)
[2, 4, 6, 8]
```

МНОЖЕСТВЕННОЕ ПРИСВАИВАНИЕ С ПОМОЩЬЮ RANGE()

- Результат range() можно использовать для присваивания нескольких переменных
- Пример одновременной инициации нескольких констант:

```
>>> SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THIRSDAY, FRIDAY, SATURDAY = range(7)
>>> SUNDAY
0
>>> MONDAY
1
>>> TUESDAY
2
>>> FRIDAY
5
>>> SATURDAY
6
```

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ. ЦИКЛ FOR

- Пример считалок с функцией range() смотрите в for_counter.py
- Пример последовательной обработки строки в цикле for смотрите в for_string.py
- Пример работы со списком в цикле for смотрите в for_list.py:
  - Почему не удались первые 2 попытки решить задачу?
  - Чем отличается перебор элементов от перебора индексов последовательности?
- Пример работы со списком, содержащим вложенные списки в циклах for смотрите в for_nested_list.py

ПРАКТИЧЕСКОЕ ЗАДАНИЕ №2.
СРАВНЕНИЕ WHILE И FOR

- Нужно посчитать сумму нечетных чисел от 1729 до 13503 включая границы
- Сделайте это с помощью цикла while и сообщите сумму
- Решите ту же задачу с помощью цикла for и проверьте результат

ПРАКТИЧЕСКОЕ ЗАДАНИЕ №3. ЦИКЛ WHILE ПЕРЕДЕЛЫВАЕМ В FOR

Поиск простых чисел до NUM_MAX:

```
>>> NUM_MAX = 20                          # верхняя граница диапазона для поиска простых чисел (константа)
>>> num = 2                               # проверяемое число будет лежать в диапазоне от num до NUM_MAX
>>> while num < NUM_MAX:                  # цикл для перебора num
        div = 2                           # проверим будет ли num делиться на div, начиная с div = 2
        while div < num:
            if num % div == 0:            # num делится на div - это число имеет целочисленный делитель
                break                     # т.е. оно составное, перейдем к следующему
            div += 1                      # это аналог присваивания div = div + 1
        else:
            print(num, "простое число") # не нашлии целочисленный делитель в цикле - это простое число
        # конец внутреннего цикла while div < num
        num += 1                          # это аналог присваивания num = num + 1
    # конец внешнего цикла while num < NUM_MAX
```

```
2 простое число
3 простое число
5 простое число
7 простое число
11 простое число
13 простое число
17 простое число
19 простое число
>>>
```

Смотрите скрипт prime_numbers.py

Что проще для этой задачи:
– while или for?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ. ПРОГРАММА «УЖАСНАЯ ПОЭЗИЯ»

- Пример на списки и цикл for для компьютерной поэзии – смотрите в awful_poetry.py

- Сам факт написания стихов компьютером дискутируется с 1950-ых годов

- Информация по этой теме:
  - https://www.ted.com/talks/oscar_schwartz_can_a_computer_write_poetry?language=ru
  - https://newtonew.com/culture/mashina-nominirovana-na-poeticheskuyu-premiyu

ПРАКТИЧЕСКОЕ ЗАДАНИЕ №4
РАЗВИТИЕ «УЖАСНОЙ ПОЭЗИИ»

- Список sys.argv[] хранит параметры запуска скрипта
- Извлеките из него 1-ый параметр
- Пусть это будет количество строк для компьютерного стихотворения
- Если параметр командной строки не задан – примите значение по умолчанию = 6

ДОМАШНЕЕ ЗАДАНИЕ. «УЖАСНАЯ ПОЭЗИЯ» V.2.0

- Исключать повторение одних и тех же частиц, существительных, глаголов и наречий в рамках каждого шестистрочия
- Пополнять списки существительных и глаголов в awful_poetry.py из файлов (*)
- См. файлы в "Scripts/AwfulData"
- Запуск скрипта должен выглядеть так:
  - awful_poetry.py <*количество_строк*> <*файл_с_существительными*> <*файл_с_глаголами*>

СПАСИБО ЗА ВНИМАНИЕ !

ВОПРОСЫ ?

School of Computer Science