DOI 10.37882/2223–2966.2021.07.32

ИСПОЛЬЗОВАНИЕ ТРАНСПАЙЛЕРА PYTHON → 11L → C++ ПРИ РЕШЕНИИ ОЛИМПИАДНЫХ ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ

USING THE PYTHON → 11L → C ++ TRANSPILER FOR COMPETITIVE PROGRAMMING

A. Tretyak
E. Vereshchagina
Yu. Dobrzhinskij
D. Zakharchenko

Summary. Nowadays Python programming language is becoming more and more popular, as well as for solving competitive programming problems. But since Python is a scripting language with dynamic typing, when solving some problems its performance is not enough and programs written in Python do not pass all the tests in the allowed time. To solve this issue it is proposed to use the Python ⟶ 11l ⟶ C++ transpiler which translates some subset of the Python programming language into C++, which significantly speeds up Python code execution (often by more than an order of magnitude). The aim of the paper is to show the specialties of this transpiler, which must be taken into account when writing Python code so that it can be correctly compiled with this transpiler. As a result, the proposed method for accelerating program code in Python can be used as a solution to the problem of insufficient performance of the original and alternative Python implementations, as a result of which there is no need to solve problems in more high-performance, but less convenient programming languages (for example, C++). An element of novelty is that the proposed transpiler in practice is not inferior in performance to the C++ language (which is not as convenient as Python in terms of syntax, including for solving competitive programming problems), unlike other similar tools for accelerating and/or executing Python code, such as Cython, PyPy, Nuitka and Shed Skin. Practical significance. This transpiler can be used to solve competitive programming problems.

Keywords: transpiler, competitive programming, programming languages.

Третьяк Александр Викторович
Аспирант, Дальневосточный федеральный университет, г. Владивосток
alextretyak2@gmail.com
Верещагина Елена Александровна
К.т.н., доцент, Дальневосточный федеральный университет, г. Владивосток
everesh@mail.ru
Добржинский Юрий Вячеславович
К.т.н., с.н.с., Дальневосточный федеральный университет, г. Владивосток
yuvd@mail.ru
Захарченко Даниил Владимирович
Аспирант, Дальневосточный федеральный университет, г. Владивосток
daniilzakharchenko@gmail.com

Аннотация. В настоящее время всё большую популярность приобретает язык программирования Python, в том числе при решении олимпиадных задач и задач по спортивному программированию. Но, так как Python является сценарным языком с динамической типизацией, то при решении некоторых задач его быстродействия недостаточно и программы, написанные на Python, не проходят все тесты в допустимое время. Для решения этой проблемы предлагается использовать транспайлер Python ⟶ 11l⟶ C++, который транслирует некоторое подмножество языка программирования Python в C++, что существенно ускоряет исполнение кода на Python (зачастую более чем на порядок). Цель работы — показать особенности использования данного транспайлера, которые необходимо учитывать при написании программного кода на языке Python, чтобы он корректно скомпилировался данным транспайлером. В результате предложенный способ ускорения программного кода на языке Python может использоваться как решение проблемы недостаточного быстродействия оригинальной и альтернативных реализаций Python, вследствие чего нет необходимости переписывать решение задачи на более высокопроизводительных, но менее удобных языках программирования (например, C++). Элементом новизны является то, что предлагаемый транспайлер на практике не уступает по производительности языку C++ (который не так удобен, как Python с точки зрения синтаксиса, в том числе для решения олимпиадных задач), в отличие от других подобных средств ускорения и/или исполнения кода на Python, которые в некоторых случаях недостаточно производительны, таких как Cython, PyPy, Nuitka и Shed Skin. Практическая значимость. Данный транспайлер может использоваться при решении олимпиадных задач и задач по спортивному программированию.

Ключевые слова: транспайлер, олимпиадное программирование, спортивное программирование, языки программирования.

Введение

В настоящее время всё большую популярность приобретает язык программирования Python (в 2019 году согласно рейтингу TIOBE [1] он обогнал по популярности язык C++), в том числе при решении олимпиадных задач и задач по спортивному программированию.

При организации большинства соревнований и олимпиад по программированию (Международная студенческая олимпиада по программированию ICPC, соревнования по спортивному программированию Codeforces, международное соревнование по программированию Google Code Jam) используются автоматические проверяющие системы для организации соревнований по программированию. Во всех этих соревнованиях действует ограничение по времени выполнения программ. Например, в задаче «Значения, которые возможно набрать» с сайта Codeforces ограничение по времени на тест составляет 2 секунды [2] и при тестировании системой решения этой задачи на языке Python возникает ошибка «превышено ограничение времени», вследствие чего данную задачу необходимо решать на более высокопроизводительном языке программирования, например, на C++. В качестве альтернативного решения, чтобы не переписывать решённую задачу с языка Python на другой язык программирования, предлагается использовать данный транспайлер.

Компилятор — это компьютерная программа, которая транслирует другие компьютерные программы для их выполнения. [3, p. 1]

Транспайлер — это такой компилятор, который переводит код из одного языка программирования в эквивалентный код на другом языке программирования. [4]

В данной статье рассматриваются особенности работы транспайлера, который переводит код на языке Python в код на новом языке программирования 11l [5], который впоследствии переводится в C++. Полученный код затем преобразуется в машинный код компилятором C++.

Следует отметить, что особенностью реализованного транспайлера является то, что он генерирует человекочитаемый код на C++, что упрощает отладку написанной программы.

Реализованный транспайлер поддерживает подмножество языка Python, достаточно большое для решения практически любых олимпиадных задач.

Фактически транспайлер Python → 11l → C++ состоит из двух транспайлеров:

1. Транспайлер Python → 11l, который переводит код на языке Python в код на новом языке программирования 11l.
2. Транспайлер 11l → C++, который переводит код на языке 11l в C++.

В некотором смысле язык 11l выступает в роли промежуточного языка, но, так как он очень похож и близок к языку Python (правда семантически, а не синтаксически), то специально изучать его, в общем-то, не требуется.

Цель данной статьи — для ускорения работы программ на языке Python использовать транспайлер Python → 11l → C++ и показать особенности его работы, которые необходимо учитывать при написании программного кода на языке Python, чтобы он корректно скомпилировался данным транспайлером.

Обзор существующих решений

Для решения задачи ускорения программ на языке Python можно использовать следующие средства ускорения и/или исполнения кода на Python, которые могут приблизить производительность полученного кода к решению задачи на языке C++, являющемся одним из самых близких к аппаратной части языков:

1. Cython
2. Nuitka
3. Shed Skin
4. PyPy

Cython — язык программирования, упрощающий написание модулей C/C++ кода для Python. Поддерживает код на чистом Python без синтаксических дополнений от Cython. Код Cython преобразуется в C/C++ код для последующей компиляции и впоследствии может использоваться как расширение стандартного Python или как независимое приложение со встроенной библиотекой выполнения Cython [6]. Cython не применяется в автоматических проверяющих системах на олимпиадах. Однако он обладает достаточно хорошей поддержкой языка Python. Значительное ускорение кода на Cython возможно только при использовании синтаксических дополнений от Cython (в том числе объявления типов, несовместимые по синтаксису с аннотациями типов в Python).

Nuitka — транспайлер, который транслирует код Python в исполняемые файлы или исходный код Си/C++. Он работает с разными версиями Python и позволяет создавать автономные приложения, даже когда Python не установлен на целевом компьютере [7]. Nuitka также,

как и Cython не применяется в автоматических проверяющих системах на олимпиадах. Nuitka практически не ускоряет, а только исполняет Python-код.

Shed Skin — транслятор программ, написанных на подмножестве языка Python, в оптимизированное C++ представление. Может транслировать только статические типы данных приложений на Python в код C++. Может генерировать как самостоятельные приложения, так и модули расширения, которые могут быть импортированы в большие приложения на языке Python [8]. Shed Skin не применяется в автоматических проверяющих системах на олимпиадах. Shed Skin поддерживает только устаревшую версию Python (2.6), а также очень ограниченное подмножество языка Python. Особенностью Shed Skin является то, что он не требует задавать в коде аннотации типов, а осуществляет вывод типов полностью автоматически. Автор использует достаточно изощрённый алгоритм, который комбинирует алгоритм декартова произведения с итеративным разделением классов. [9]

PyPy — интерпретатор языка программирования Python. PyPy в начале своего существования был интерпретатором Python, написанным на Python. Текущие версии PyPy транслируются из RPython в Си и компилируются. В PyPy встроен трассирующий JIT-компилятор, который может превращать код на Python в машинный код во время выполнения программы [10]. PyPy — единственный из рассматриваемых средств для ускорения кода на Python, который применяется в автоматических проверяющих системах на олимпиадах. Он обладает очень хорошей поддержкой Python современных версий, однако согласно проведенным тестам PyPy по производительности уступает Shed Skin и предлагаемому транспайлеру Python ⟶ 11l ⟶ C++.

Python ⟶ 11l ⟶ C++ — транспайлер, переводящий код на языке Python в код на языке C++, за счёт чего производится значительное ускорение исходной программы. В отличие от Shed Skin данный транспайлер не использует сложных алгоритмов для автоматического вывода типов, а использует стандартный синтаксис аннотаций типов языка Python. По сравнению со всеми вышеперечисленными он обладает самой высокой производительностью и практически не уступает по скорости выполнения коду, написанному на C++ вручную.

Далее предлагается рассмотреть такие особенности транспайлера, как целые числа, символьные переменные, создание пустого списка/массива, словаря и множества, создание непустого списка/массива, словаря и множества, передача списка как аргумента функции, переменные-члены классов, скобки у кортежей, конкатенация строк, строковые и символьные литералы, множественная инициализация, конструкция from … import …, рекурсивный вызов функции, обход элементов словаря в цикле for, деление, остаток от деления, конструкция yield, порядок вычисления аргументов функции, передача функции в качестве аргумента другой функции, конструкция [… if … else … for …], которые необходимо учитывать при написании программного кода на языке Python, чтобы он корректно скомпилировался данным транспайлером. Следует отметить, что возможность объявлять символьные переменные может использоваться для увеличения производительности и для уменьшения занимаемой оперативной памяти.

Целые числа

Как известно, в языке программирования Python все целые числа произвольного размера. С одной стороны, это позволяет программисту не задумываться о том, насколько большое число может потребоваться сохранить в переменной целого типа. Но с другой стороны работа с такими числами гораздо более ресурсоёмкая и менее эффективная. Поэтому все целые числа транспайлер Python ⟶ 11l ⟶ C++ рассматривает (как и принято в C++ по умолчанию) как 32-разрядные целые. Если требуется работать с числами большей разрядности (например, 64), то можно либо явно указать тип переменной как Int64 либо использовать опцию транспайлера —int64 в командной строке (в этом случае все целые числа будут рассматриваться не как 32-, а как 64-разрядные). Явное указание типа переменной выглядит так:

```
s: Int64 = 0
```

Но в Python-е нет встроенного типа Int64. Поэтому, если требуется, чтобы код работал в Python, необходимо в начале программы один раз добавить такую строчку:

```
Int64 = int
```

Но транспайлер Python ⟶ 11l ⟶ C++ поймёт исходный Python-код и без такой строчки (а если встретит такую строку, то просто проигнорирует её).

Символьные переменные

Аналогично целым 64-разрядным числам, транспайлер Python ⟶ 11l ⟶ C++ предоставляет возможность объявлять символьные переменные, то есть переменные, значением которых является код одного символа. Тип символьной переменной — Char. И аналогично Int64, чтобы получить код, который работает в Python, следует добавить такую строчку:

Таблица 1. Код для создания пустого списка, словаря и множества

Не поддерживается	Поддерживается
l = []	l: List[int] = [] или l = [0]*0
d = {}	d: Dict[str, int] = {}
s = set()	s = set() # int
dd = collections.defaultdict(int)	dd = collections.defaultdict(int) # str

Таблица 2. Код объявления нового класса

Не скомпилируется	Скомпилируется

class Error(Exception):
def __init__(self, message, pos):
self.message = message
self.pos = pos
	class Error(Exception):
	message: str
	pos: int
	def __init__(self, message, pos):
	self.message = message
	self.pos = pos

Char = str

Объявить массив из символов можно так:

charr: List[Char] = []

Это может использоваться как для увеличения производительности, так и для уменьшения занимаемой оперативной памяти (особенно если необходим очень большой массив символов, например).

Создание пустого списка/массива, словаря и множества

В отличие от языка Python, контейнеры (например списки и массивы) в языках 11l и C++ могут содержать элементы только одного типа (за исключением кортежей, которые могут содержать элементы различных типов), причем известного во время компиляции. На данном этапе транспайлер Python ⟶ 11l ⟶ C++ не пытается определить тип контейнеров по их использованию, как это сделано в некоторых языках программирования (например Nemerle [11]), поэтому при создании пустого контейнера необходимо указать тип его элементов явно (см. табл. 1).

Если типом значения словаря collections.defaultdict является список, то необходимо использовать следующую форму записи:

dd: DefaultDict[str, List[int]] = collections.defaultdict(list)

Создание непустого списка/массива, словаря и множества

Если контейнер инициализируется элементами, то указывать его тип не требуется:

l = [1, 2, 3]
d = {'a': 1, 'b': 2}
s = {1, 2, 3}

Так как все элементы списков должны быть одного типа, то такой список не скомпилируется:

x = [0, 1, 2, 2, 2, 2, 1, 9, 3.5, 5, 8, 4, 7, 0, 6]

Чтобы это исправить достаточно указать тип первого элемента:

x = [float(0), 1, 2, 2, 2, 2, 1, 9, 3.5, 5, 8, 4, 7, 0, 6]

Передача списка как аргумента функции

В Python при передаче переменной типа list в функцию, её можно изменять внутри функции. И чтобы возможность изменять переменную сохранилась, транспайлеру Python ⟶ 11l ⟶ C++ необходимо явно указать тип этого аргумента.

Так, при объявлении функции def decompress(compressed) в коде-решении задачи LZW compression [12] следует указать тип аргумента:

```
def decompress(compressed: List[int]):
…
```

Или так:

```
def decompress(compressed: list):
```

Переменные-члены классов

Чтобы код объявления нового класса успешно скомпилировался транспайлером Python ⟶ 11l ⟶ C++ необходимо указать типы всех переменных-членов этого класса (см. табл. 2).

Скобки у кортежей

В языке Python в некоторых случаях допускается опускать скобки у кортежей. Например, можно писать return a, b вместо return (a, b), или a, b = b, a вместо (a, b) = (b, a). В то время как в Python поддерживаются обе эти записи, транспайлер Python ⟶ 11l ⟶ C++ поддерживает только вариант со скобками.

Конкатенация строк

Так как язык программирования 11l не допускает использование оператора + для конкатенации строк (по причинам, обозначенным в документации языка [13]), а использует свой синтаксис для данной операции, то транспайлер Python ⟶ 11l пытается угадать в каких случаях оператор + является арифметическим, а в каких является оператором конкатенации строк. В случае если ему не удалось правильно определить конкатенацию строк, следует между операндами добавить прибавление пустой строки (вместо str1 + str2 следует написать str1 + '' + str2), например:

```
aa = ['1', '2']
bb = ['x', 'y']
for a in aa:
for b in bb:
print(a + '' + b)
```

Однако на практике в большинстве случаев, когда транспайлеру Python ⟶ 11l не удалось определить конкатенацию строк, достаточно добавить аннотацию типа. Например, в таком коде:

```
def rotated(s):
return s[1:] + s[0]
```

достаточно указать тип аргумента s:

```
def rotated(s : str):
return s[1:] + s[0]
```

(Вместо того чтобы писать s[1:] + '' + s[0]).

Строковые и символьные литералы

Так как тип выражения «A» в языке 11l зависит от контекста (он может быть либо String, либо Char), в случае неверного определения типа строкового литерала транспайлером необходимо указать тип явно, т.е. писать str('A') либо Char('A'). Так, следующий Python-код не скомпилируется:

```
print(['AF'] + ['A']*5)
```

И необходимо писать так:

```
print(['AF'] + [str('A')]*5)
```

Также, если есть переменная типа Char (ch : Char), то чтобы присвоить ей символ необходимо написать: ch = Char('*') вместо ch = '*'.

Множественная инициализация

На данный момент транспайлер Python ⟶ 11l ⟶ C++ не поддерживает множественную инициализацию, и вместо a = b = 0 следует писать:

```
a = 0
b = 0
```

(Однако запись if a == b == c …, а также if a < b < c … поддерживается.)

Поддерживаемые модули

На данный момент транспайлер Python ⟶ 11l ⟶ C++ поддерживает следующие встроенные модули Python:

- math
- os
- time
- re
- random
- collections (только defaultdict и deque)
- heapq

Конструкция from … import …

На данный момент транспайлер Python ⟶ 11l ⟶ C++ не поддерживает данную конструкцию, и вместо такого кода:

```
from math import sqrt
print(sqrt(2))
```

следует писать:

```
import math
print(math.sqrt(2))
```

Рекурсивный вызов функции

Если функция вызывает сама себя [рекурсивно], то при её объявлении необходимо явно указать тип возвращаемого значения (так как он не может быть выведен автоматически компилятором C++ в этом случае).

```
def find(x, y) -> None: # `-> None` здесь обязательно
find(nx, ny) # из-за этого вызова
```

В случае рекурсивного вызова локальной функции требуется указывать не только тип возвращаемого значения, но и типы всех аргументов функции:

```
def fib(n):
def f(n : int) -> int: # написать просто `def f(n):`
# или `def f(n) -> int:` нельзя
if n < 2:
return n
return f(n-1) + f(n-2)
return f(n)
```

Это связано с особенностью работы компилятора C++.

Обход элементов словаря в цикле for

При обходе словаря в цикле for в том случае, когда транспайлер Python ⟶ 11l не смог определить, что итерируемый контейнер — это словарь, следует указать явно, что необходимо обходить словарь по ключам, а не по парам (ключ, значение):

```
for k in d: # если не удалось определить тип `d`,
print(k) # то не скомпилируется

# И в таком случае необходимо писать так:

for k in d.keys():
print(k)
```

Деление

Если делитель и делимое не являются числовыми литералами и при этом являются целочисленными, то деление выполняется по правилам C++ и Python 2, а не Python 3, т.е. деление в таком случае целочисленное. Чтобы получить вещественное деление необходимо использовать приведение к вещественному типу (т.е. вместо a/b написать float(a)/b или a/float(b)), либо использовать опцию транспайлера —python-division, чтобы операция / всегда выполнялась по правилам Python 3.

Остаток от деления

Если a может быть меньше 0, то вместо a% b следует писать:

```
((a % b) + b)% b
или
r = a% b; if (r < 0) r += b
```

Так как операция% выполняется по правилам C++, а не Python.

Либо можно использовать опцию транспайлера —python-remainder, чтобы операция% выполнялась по правилам Python.

Конструкция yield

На данный момент транспайлер Python ⟶ 11l ⟶ C++ не поддерживает данную конструкцию, и вместо такого кода:

```
def squares(n):
for i in range(n):
yield i ** 2
```

следует писать:

```
def squares(n):
r: List[int] = []
for i in range(n):
r.append(i ** 2)
return r
```

Порядок вычисления аргументов функции

В отличие от языка Python в языке C++, к сожалению, порядок вычисления аргументов при передаче в функцию не определён, поэтому необходимо следить за возможной неправильной работой кода, производящего вычисления во время передачи параметров функции.

Вот строчка кода из примера реализации калькулятора обратной польской записи на языке Python [14]:

```
a.append(b[c](a.pop(), a.pop()))
```

Чтобы она корректно работала с транспайлером Python ⟶ 11l ⟶ C++, её необходимо переписать например так:

```
t = a.pop()
a.append(b[c](t, a.pop()))
```

Передача функции в качестве аргумента другой функции

Вот код решения задачи Sort using a custom comparator [15]:

```
def mykey(x):
return (–len(x), x.upper())
print(sorted(strings, key=mykey))
```

Оннескомпилируетсятранспайлером Python ⟶ 11l ⟶ C++, так как в сгенерированном C++ коде mykey — это шаблонная функция.

Следует либо указать типы аргументов функции:

```
def mykey(x : str):
return (–len(x), x.upper())
```

Либо использовать лямбда-функцию:

```
mykey = lambda x: (–len(x), x.upper())
```

Конструкция [... if ... else ... for ...]

Такой код не скомпилируется:

```
l = [b if char == "w" else 1 for char in input()]
```

Необходимо добавить скобки:

```
l = [(b if char == "w" else 1) for char in input()]
```

Производительность

Ниже приводится код на Python — решение задачи нахождения k-ого по счёту простого числа [16].

```
import math
k = int(input())
n = k * 17
primes = [True] * n
primes[0] = primes[1] = False
for i in range(2, int(math.sqrt(n)) + 1):
if not primes[i]:
continue
for j in range(i * i, n, i):
primes[j] = False
for i in range(n):
if primes[i]:
if k == 1:
print(i)
break
k -= 1
```

Для k равного 1000000 время работы данной программы составило 20 секунд используя эталонную реализацию Python (т.е. CPython), такое же время выполнения у Nuitka, 7.4 секунды используя Cython, 2.7 секунд используя PyPy, 1.4 секунды используя Shed Skin и 0.73 секунды используя транспайлер Python ⟶ 11l ⟶ C++.

Заключение

В данной статье были рассмотрены основные особенности транспайлера Python ⟶ 11l ⟶ C++: создание пустого списка, передача списка как аргумента функции, переменные-члены классов, конкатенация строк, рекурсивный вызов функции.

На первый взгляд эти особенности усложняют работу с ним (в идеале хорошо бы иметь возможность компилировать абсолютно любой код на Python, и в таком случае никаких правил, представленных в данной статье, описывать бы вообще не пришлось). Отчасти это действительно так, однако на практике это не приводит к скрытым ошибкам, так как все (за редким исключением) описанные рекомендации в случае их несоблюдения приведут к ошибкам на этапе компиляции, в то время как в языке Python нюансы его работы приводят к реальным трудно уловимым ошибкам, например в Python для создания двумерного массива размера n на m вместо [[0] * m] * n необходимо писать [[0] * m for i in range(n)], кроме того, необходимо следить за именами переменных циклов (например, цикл по переменной i не следует вкладывать внутрь другого цикла по переменной i, т.к. в Python-е нет областей видимости локальных переменных), а также нельзя так просто добавить список или другой объект как значение по умолчанию аргумента функции (например запись def f(l = []) приводит к нежелательному поведению на практике).

Ускорение программного кода происходит за счёт того, что в отличие от CPython, который является интерпретатором байт-кода, транспайлер Python ⟶ 11l ⟶ C++ генерирует C++ код, который компилируется непосредственно в машинный код.

В итоге реализованный транспайлер [17] вполне пригоден для добавления его поддержки в автоматические проверяющие системы на олимпиадах и сайтах по спортивному программированию наряду с компилятором PyPy, который хоть и уступает транспайлеру Python ⟶ 11l ⟶ C++ по производительности, но обладает лучшей поддержкой языка Python.

ЛИТЕРАТУРА

1. TIOBE Index | TIOBE — The Software Quality Company [Электронный ресурс]. — URL: https://www.tiobe.com/tiobe-index/ (дата обращения: 16.04.2021)
2. Задача — 687C — Codeforces [Электронный ресурс]. — URL: https://codeforces.com/problemset/problem/687/C (дата обращения: 16.04.2021)
3. Keith Cooper, Linda Torczon. Engineering: A Compiler (2nd Edition) — Morgan Kaufmann, USA, 2012. — 824 p.
4. Транспайлер — Википедия [Электронный ресурс]. — URL: https://ru.wikipedia.org/wiki/Транспайлер (дата обращения: 16.04.2021)
5. Третьяк А.В., Третьяк Е.В., Верещагина Е.А. Разработка когнитивно-эргономического синтаксиса для нового аппаратно-ориентированного языка программирования // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и Технические Науки. — 2020. — № 07. — С. 145–153 DOI 10.37882/2223–2966.2020.07.33
6. Cython — Википедия [Электронный ресурс]. — URL: https://ru.wikipedia.org/wiki/Cython (дата обращения: 16.04.2021)
7. Nuitka — Википедия [Электронный ресурс]. — URL: https://ru.wikipedia.org/wiki/Nuitka (дата обращения: 16.04.2021)
8. Shedskin — Википедия [Электронный ресурс]. — URL: https://ru.wikipedia.org/wiki/Shedskin (дата обращения: 16.04.2021)
9. Mark Dufour. Shed Skin. An Optimizing Python-to-C++ Compiler [Текст]: дис.маг./ Делфтский технический университет, Committee: Dr.ir. K.G. Langendoen, Prof.dr.ir. H.J. Sips, Dr.ing. L.M.F. Moonen — Делфт, Нидерланды: 2006, 57 с. — Библиогр.: 13–14 с.
10. PyPy — Википедия [Электронный ресурс]. — URL: https://ru.wikipedia.org/wiki/PyPy (дата обращения: 16.04.2021)
11. About — Nemerle programming language official site [Электронный ресурс]. — URL: http://nemerle.org/About (дата обращения: 16.04.2021)
12. LZW compression [Электронный ресурс]. — URL: https://www.rosettacode.org/wiki/LZW_compression#Python (дата обращения: 16.04.2021)
13. Документация языка программирования 11l [Электронный ресурс]. — URL: http://11l-lang.org/doc/ru/%E2%80%98%E2%80%99 (дата обращения: 16.04.2021)
14. RPN calculator algorithm [Электронный ресурс]. — URL: https://www.rosettacode.org/wiki/Parsing/RPN_calculator_algorithm#Python (дата обращения: 16.04.2021)
15. Sort using a custom comparator [Электронный ресурс]. — URL: https://www.rosettacode.org/wiki/Sort_using_a_custom_comparator#Python (дата обращения: 16.04.2021)
16. Задача № 973. Простое число [Электронный ресурс]. — URL: https://informatics.msk.ru/mod/statements/view.php?chapterid=973 (дата обращения: 16.04.2021)
17. Язык программирования 11l [Электронный ресурс]. — URL: http://11l-lang.org/ru (дата обращения: 16.04.2021)