DOI: 10.32517/2221-1993-2023-22-1-7-12

К. И. Кольцова
победитель конкурса ИНФО-2022 в номинации «Инновации в методике обучения информатике», Парфеньевская средняя общеобразовательная школа, с. Парфеньево, Костромская область, Россия; Ярославский государственный педагогический университет им. К. Д. Ушинского, г. Ярославль, Россия

ИСПОЛЬЗОВАНИЕ СЮЖЕТНЫХ ЗАДАЧ ПРИ ОБУЧЕНИИ ПРОГРАММИРОВАНИЮ НА PYTHON

Аннотация
В статье рассматривается проблема повышения внутренней мотивации школьников при изучении программирования. К традиционным приемам, способствующим повышению мотивации, можно отнести апелляцию к жизненному опыту учащихся, создание проблемной ситуации, решение сюжетных задач, моделирование в учебном процессе реальных событий (ролевые игры). Одним из требований, предъявляемых к задачам с прикладным содержанием, является соответствие фабулы возрастным особенностям учащихся. Выполнение этого требования будет возможным, если ситуация, описанная в фабуле задачи, социально значима для подростков, учитывает их интересы и современные тенденции подростковой коммуникации. В статье приведены примеры сюжетных задач, использование которых целесообразно при обучении программированию учащихся основной школы. Указанные задачи позволяют познакомить учащихся со структурами данных «список» и «словарь», с генератором псевдослучайных чисел и с созданием собственных модулей на Python.
Ключевые слова: мотивация, мотив, язык программирования Python, сюжетная задача.

1. Введение

Одной из центральных проблем обучения является проблема мотивации обучающихся. Иногда на уроках можно услышать от учеников фразу: «Зачем это нужно? Мне в жизни это не пригодится». Актуальность обозначенной проблемы обусловлена также требованиями профессиональных стандартов, предъявляемыми педагогам, обновлением содержания обучения и самой учебной деятельностью [10, 11]. В ФГОС нового поколения подчеркивается необходимость создания условий, которые будут способствовать формированию и развитию мотивации по освоению учебной дисциплины.

Исследованию природы мотивации и мотивов посвящено множество трудов как зарубежных, так и отечественных ученых: Л. И. Божович, А. Н. Леонтьева, П. В. Симонова, К. К. Платонова, К. Мадсен, А. Маслоу и др. Существует большое разнообразие трактовок понятия ***«мотивация»***. Например, в статье [8] мотивация определяется как осмысление индивидом ситуации, выбор и оценка всевозможных моделей поведения, их предполагаемых итогов и формирование на данной базе мотивов. В «Словаре практического психолога» С. Ю. Головина [3] мотивация определяется как система побуждений, вызывающие активность организма и определяющие ее направленность; осознаваемые

Контактная информация

Кольцова Ксения Игоревна,
учитель физики, Парфеньевская средняя общеобразовательная школа, с. Парфеньево, Костромская область, Россия; *адрес:* 157270, Россия, Костромская область, с. Парфеньево, ул. Маркова, д. 17;
магистрант кафедры теории и методики обучения информатике, физико-математический факультет, Ярославский государственный педагогический университет им. К. Д. Ушинского, г. Ярославль, Россия; *адрес:* 150000, Россия, г. Ярославль, ул. Республиканская, д. 108/1;
e-mail: study.zarubina@yandex.ru

K. I. Koltsova
The Parfenyevo Secondary School, Parfenyevo, Kostroma Region, Russia;
Yaroslavl State Pedagogical University named after K. D. Ushinsky, Yaroslavl, Russia

USING PLOT TASKS WHEN TEACHING PYTHON PROGRAMMING

Abstract
The article deals with the problem of increasing the internal motivation of schoolchildren when studying programming. The traditional techniques that increase motivation include appealing to the life experience of students, creating a problem situation, solving plot tasks, modeling real events in the educational process (role-playing games). One of the requirements for tasks with applied content is the correspondence of the plot to the age characteristics of students. The fulfillment of this requirement will be possible if the situation described in the plot of the task is socially significant for adolescents, takes into account their interests and current trends in adolescent communication. The article provides examples of plot tasks, the use of which is expedient when teaching programming to basic school students. These tasks allow students to get acquainted with the "list" and "dictionary" data structures, with a pseudo-random number generator, and with creating their own modules in Python.
Keywords: motivation, motive, Python programming language, plot task.

или неосознаваемые психические факторы, побуждающие индивида к совершению определенных действий и определяющие их направленность и цели. Р. С. Немов пишет, что мотивация — это совокупность причин психологического характера, которые объясняют поведение человека, а точнее, его начало, активность и направленность [7]. К. К. Платонов считает, что мотивация — это совокупность мотивов [9]. Ж. Годфруа определяет мотив как соображение, по которому субъект должен действовать [2]. По Е. П. Ильину, мотив — это субъективная причина (осознанная или неосознанная) того или иного поведения, действия человека — психическое явление, непосредственно побуждающее человека к выбору способа действия и его осуществления [6].

Существует ряд приемов, способствующих повышению учебной мотивации:

- апелляция к жизненному опыту учащихся;
- создание проблемной ситуации;
- решение сюжетных задач;
- моделирование в учебном процессе реальных событий (ролевые игры).

Под ***сюжетной задачей*** Л. М. Фридман понимает задачи, в которых описан некоторый жизненный сюжет с целью нахождения определенных характеристик или значений [12].

Согласно М. В. Егуповой, одним из требований, предъявляемых к задачам с прикладным содержанием, является соответствие фабулы возрастным особенностям учащихся [5]. Выполнение этих требований будет возможным, если:

- ситуация, описанная в фабуле задачи, будет понятна учащимся;
- будут учтены современные тенденции подростковой коммуникации;
- содержание задачи ориентировано на социально значимые для подростков ситуации;
- будут учтены интересы и хобби подростков.

В данной статье приведены примеры сюжетных задач, которые апеллируют к жизненному опыту учащихся и могут быть использованы для создания проблемной ситуации на уроке при изучении основ программирования на языке Python.

2. Пример сюжетной задачи для знакомства учащихся со структурой данных «список» — задача «Странник»

Представьте, что вы являетесь героем компьютерной игры. Предположим, вы странник, а у такого персонажа всегда есть сумка для припасов и лекарств. Давайте подумаем и выпишем на доску, что будет лежать в вашей сумке.

Ребята, как мы можем представить список предметов в программе? Давайте представим его в виде строки, поместив ее в переменную *bag* (*англ.* сумка):

```
bag="бутыль воды, ломоть хлеба, чеснок, спирт,
кусок тряпки, подорожник"
```

Итак, мы создали строку. Напомните мне: какие операции со строками вы помните? (*Учащиеся называют операции: сложение, дублирование, нахождение элемента по индексу, длина строки.*) [4]

Теперь представьте ситуацию: вы, странник, случайно забрели на развалины замка и, покопавшись в груде камней, нашли клинок. Взяли его в руку и решили припрятать на всякий случай в сумку. Обозначим клинок как *knife* (*англ.* нож). Как мы сможем добавить этот элемент к строке (т. е. положить в сумку)? Давайте посчитаем, сколько теперь предметов в сумке:

```
knife="нож"
bag1=knife+bag
print(bag1)
print("Количество элементов в сумке: ",
len(bag1))
```

Запустим программу и увидим, что она выведет:

```
ножбутыль воды, ломоть хлеба, чеснок, спирт,
 кусок тряпки, подорожник
Количество элементов в сумке: 69
```

Внимательно посмотрите на результат. Вы получили то, что хотели найти?

В этом случае лучше воспользоваться более удобным объектом. В Python существует объект, который называется «список».

Чтобы создать список, нужно написать переменную и присвоить ей элементы, стоящие в квадратных скобках через запятую.

А теперь вернемся к задаче.

Создадим изначальный список вещей в сумке:

```
bag=["бутыль воды", "ломоть хлеба", "чеснок",
 "спирт", "кусок тряпки", "подорожник"]
```

Мы можем обратиться к любому элементу списка, указав в квадратных скобках его позицию.

Давайте попробуем достать из сумки кусок тряпки. Помните, что отсчет начинается с 0. Значит, тряпка у нас находится в списке под номером 4:

```
print(bag[4])
```

А сейчас настало время добавить в сумку новый элемент — нож. Сделаем это с помощью функции *append*():

```
bag.append("нож")
```

Молодцы, теперь проверим, что есть в сумке:

```
print(bag)
```

Время идет, странник уже слишком долго шел, пора бы остановиться, подкрепиться и лечь спать. Вы попили воды и съели весь хлеб. Перед нами новая задача: как удалить элемент из сумки?

Для этой задачи применим ключевое слово *del* к нужному нам элементу:

```
del bag[1]
```

Новый день, новые путешествия. Пора бы пополнить запасы еды! Вы идете на базар и выбираете любимые лакомства: окорочок, свиные ушки и бутылку эля. Итак, появился новый список, давайте его поместим в переменную *food*.

```
food=["окорочок","свиные ушки","эль"]
```

А теперь положим новоприобретение в сумку. Ничего сложного, все так же, как и сложение строк.

Обязательно стоит проверить продукты в сумке, вдруг что-то выронили.

```
bag=bag+food
```

Хм, что-то сумка стала уж больно тяжела. Не порвется ли она? Мы можем поместить в сумку лишь 15 предметов. Давайте узнаем, сколько уже есть. Воспользуемся функцией *len*():

```
print("Количество продуктов в сумке: ", len(bag))
```

Уже девять элементов! Нужно избирательнее относиться к предметам, которые мы кладем в сумку.

Пора бы уходить с базара, но не тут-то было — вы вдруг вспомнили, что с окорочком хорошо бы взять и чеснок. А вы и не помните, есть ли он в сумке. Давайте напишем код, который будет проверять это. Если предмет, который бы вы хотели взять, уже есть в сумке, то вам будет выводиться предупреждение об этом. Проверку мы можем осуществить, применив условную конструкцию *if* и оператор *in*:

```
if n in bag:
    print("У тебя уже есть в сумке ", n)
else:
    print("В твою сумку может поместиться
     15 предметов. А в ней уже ", len(bag),
     "предметов. Точно хочешь взять ", n, " ?")
```

Итак, проверив все, вы обнаружили, что в чесноке нет нужды. И пошли дальше бродить по полям и тропам...

3. Пример сюжетной задачи для знакомства учащихся со структурой данных «словарь» — задача «Словарь сленга»

Современный молодежный мир породил множество новых сленговых слов. Зачастую, общаясь с родителями, по привычке вы их используете. Но вот проблема! Вас не понимают. А давайте попробуем разрешить это затруднение! Возьмем и создадим для наших пап и мам толковый словарь молодежного сленга.

Мы уже с вами знаем, что в Python существуют последовательности вида «список» и «кортеж». Кортеж неизменяем, такое нам не подойдет. Список можно менять, удалять из него и добавлять новые элементы, но он состоит из набора значений. Нам же нужно, чтобы был набор пар. Объект, который нам в этом случае поможет, называется «словарь».

Словарь состоит из набора элементов «ключ: значение», которые разделены запятой и заключены в фигурные скобки. Ключи не должны повторяться, в отличие от значений (в значениях также могут быть и разные типы данных). А теперь посмотрим на примере, как выглядит словарь:

```
dictionary = {1: "Adam", 2: "Anna"}
```

Здесь:

- *dictionary* — название нашего словаря;
- 1, 2 — ключи;
- Adam, Anna — значения.

Чтобы вывести содержимое нашего словаря на экран, достаточно использовать функцию *print*().

А теперь подумаем над тем, какие термины мы бы хотели разъяснить родителям (*выписываем на доску термины, которые предложили дети*).

Ребята, для понимания программы ограничимся четырьмя наборами. Составим с ними словарь:

```
dictionary = {"хайпануть": "привлечь к себе
 внимание, стать известным", "хайп": "то, что
 привлекает к себе внимание, шумиха", "чилить":
 "прохлаждаться, расслабляться, ничего не
 делать", "поймал флешбэки": "на меня накатили
 воспоминания"}
```

Как вы думаете, какие возможности могут быть у словаря? (*Ответы учащихся*.)

Итак, со словарем можно выполнять следующие операции:

1) найти толкование;
2) добавить новый термин;
3) удалить термин;
4) изменить значение ключа;
5) очистить словарь;
6) выйти из программы.

Напишем код меню и сразу добавим разветвление:

```
print("Добро пожаловать в словарь молодежного
 сленга!")
print("\t\t\tМеню\n0. Найти толкование\n1.
 Добавить новый термин\n2. Удалить термин\n3.
 Изменить значение ключа\n4. Очистить
 словарь\n5. Выйти из программы")
number=int(input("Выберите действие. Напишите
 номер: "))
if number==0:
    print("\t\t\tТолкование какого термина
     вы бы хотели узнать?")
    term=input("Введите термин: ")
elif number==1:
    print("\t\t\tКакой термин вы бы хотели
     добавить?")
    term=input("Введите термин: ")
elif number==2:
    print("\t\t\tКакой термин вы бы хотели
     удалить?")
    term=input("Введите термин: ")
elif number==3:
    print("\t\t\tЗначение какого термина вы бы
     хотели заменить?")
    term=input("Введите термин: ")
elif number==4:
    print("\t\t\tВы точно хотите очистить
     словарь? ")
elif number==5:
    print("\t\t\tЧтобы выйти из программы,
     нажмите Enter")
else:
    print("Такого пункта нет в меню")
```

Поговорим обо всем по порядку.

1. Получение доступа к значению в словаре.

Самый легкий и прямой способ получить доступ к значению в словаре — это ввод имени словаря и ключа в квадратных скобках, т. е. обращение напрямую.

Не забывайте о том, что Python является регистрочувствительным!

```
print(dictionary["хайп"])
```

Получается, что при введении ключа мы как бы открываем ящик, в котором содержится толкование термина.

Внимательный слушатель скажет, что это похоже на доступ к элементу списка по индексу. Но это не совсем так. Если в списке мы используем индекс, то в словаре — ключ. Но если мы с вами захотим обратиться к элементу словаря посредством индекса, то эта идея ничего полезного нам не даст. И вот почему: в словаре не важен порядок, в котором идут пары, а следовательно, индексации в принципе нет, как в других последовательностях, изученных нами ранее.

Добавим все необходимое в наш код:

```
print("Толкование какого термина вы бы хотели
 узнать?")
term=input("Введите термин: ")
print("это значит", dictionary[term])
```

Здесь *term* — это переменная, которая содержит введенный термин.

Опробуем приведенную программу. Работает? А теперь попробуем ввести термин «пинг», которого у нас нет в словаре. Внимание, появилась ошибка!

```
Толкование какого термина вы бы хотели узнать?
Введите термин: пинг
KeyError: 'пинг'
```

KeyError появляется тогда, когда Python проверил весь словарь, но не нашел введенного нами ключа. То есть интерпретатор спешит сообщить нам, что проблема — в ключе.

Необходимо избавиться от этой ошибки. Получается, что программа сначала должна проверить, а есть ли термин в нашем словаре. В случае его ненахождения, нужно обязательно об этом оповестить пользователя. Воспользуемся уже известным оператором *in*:

```
if term in dictionary:
    print("это значит", dictionary[term])
else:
    print("Извините, этого термина еще нет
     в словаре")
```

2. Добавление нового термина.

Словарь — это мутирующий объект. То есть его элементы мы можем удалять и добавлять новые. Сначала разберемся, как же добавить новую пару.

```
dictionary["вайб"]="синоним атмосферы,
 душевного состояния, энергетики"
```

Мы пишем имя словаря, в который хотим добавить элемент, в квадратных скобках пишем ключ, ставим «=» и пишем значение.

Стоит также отметить, что если вы будете использовать уже занятый под значение ключ, то старое значение будет стерто, а на его месте будет новое. Поэтому нам следует написать небольшой код, который будет проверять, есть ли уже этот ключ у нас в словаре. В этот раз нам понадобится оператор *not*:

```
term=input("Напишите, какой термин вы бы хотели
 добавить: ")
if term not in dictionary:
    interpretation=input("Введите толкование
     вашего термина: ")
    dictionary[term]=interpretation
    print("Ваш термин был добавлен в словарь")
else:
    print("У этого ключа уже есть значение!",
     term, " - это", dictionary[term])
```

Разберемся со строкой:

```
dictionary[term]=interpretation
```

Здесь:
dictionary — название нашего словаря;
term — сам термин;
interpretation — значение нашего термина.

3. Удаление термина из словаря.

Удаление выполняется с помощью знакомой нам инструкции *del*. Покажем на примере, как это осуществить. Не забываем о проверке наличия термина в нашем словаре.

```
term=input("Напишите, какой термин вы бы хотели
 удалить: ")
if term in dictionary:
    del dictionary[term]
    print("Термин", term, "удален")
else:
    print("Такого термина нет в нашем словаре")
```

4. Переопределение термина.

Чтобы заменить значение ключа, достаточно присвоить новое определение, и старое будет удалено.

```
term=input("Напишите, значение какого термина
 вы бы хотели переопределить: ")
if term in dictionary:
    interpretation=input("Введите толкование
     вашего термина: ")
    dictionary[term]=interpretation
    print("Ваш термин переопределен")
else:
    print("Такого термина еще нет в словаре.
     Если хотите его добавить, вернитесь в меню")
```

5. Удаление содержимого словаря.

Если, переосмыслив всё, вам захочется заново переписать словарь, можно очистить содержимое с помощью метода *clear*(), применив его к названию нашего словаря:

```
answer=input("Вы точно хотите очистить
 словарь? ")
if answer=="Да" or answer=="да":
    dictionary.clear()
    print("Словарь пуст")
else:
    print("Спасибо, что не сделали этого")
```

6. Выход из программы.

```
input("Чтобы выйти из программы, нажмите Enter")
```

Поздравляю, все части программы были проанализированы, соберем все воедино.

```
dictionary = {"хайпануть": "привлечь к себе
 внимание, стать известным ","хайп" : "то, что
 привлекает к себе внимание, шумиха", "чилить" :
 "прохлаждаться, расслабляться, ничего не
 делать", "поймал флешбэки" : "на меня накатили
 воспоминания"}
print("Добро пожаловать в словарь молодежного
 сленга!")
print("\t\t\tМеню\n0. Найти толкование\n1.
 Добавить новый термин\n2. Удалить термин\n3.
 Изменить значение ключа\n4. Очистить
 словарь\n5. Выйти из программы")
number=int(input("Выберите действие. Напишите
 номер: "))
if number==0:
    print("Толкование какого термина вы бы
 хотели узнать?")
    term=input("Введите термин: ")
    if term in dictionary:
        print("это значит", dictionary[term])
    else:
        print("Просим прощения, этого термина
         еще нет в словаре")
elif number==1:
    term=input("Напишите, какой термин вы бы
     хотели добавить: ")
    if term not in dictionary:
        interpretation=input("Введите
         толкование вашего термина: ")
        dictionary[term]=interpretation
        print("Ваш термин был добавлен
         в словарь")
    else:
        print("У этого ключа уже есть
         значение!", term, " - это",
         dictionary[term])
elif number==2:
    term=input("Напишите, какой термин вы бы
     хотели удалить: ")
    if term in dictionary:
        del dictionary[term]
        print("Термин", term, "удален")
    else:
        print("Такого термина нет в нашем
         словаре")
elif number==3:
    term=input("Напишите, значение какого
     термина вы бы хотели переопределить: ")
    if term in dictionary:
        interpretation=input("Введите
         толкование вашего термина: ")
        dictionary[term]=interpretation
        print("Ваш термин переопределен")
    else:
        print("Такого термина еще нет
         в словаре. Если хотите его добавить,
         вернитесь в меню")
elif number==4:
    answer=input("Вы точно хотите очистить
     словарь? ")
    if answer=="Да" or answer=="да":
        dictionary.clear()
        print("Словарь пуст")
    else:
        print("Спасибо, что не сделали этого")
elif number==5:
    input("Чтобы выйти из программы, нажмите
     Enter")
else:
    print("Такого пункта нет в меню")
```

Этот код можно бесконечно дополнять, по своему желанию вы можете добавить пароль на некоторые пункты меню, которые хотели бы спрятать от других. Так что придумывайте и дополняйте!

4. Пример сюжетной задачи для знакомства учащихся с генератором псевдослучайных чисел — задача «Аффирмация дня»

Мы всю свою жизнь занимаемся самовнушением и, сами того не замечая, чаще всего внушаем себе что-то нехорошее. Например: «Я не смогу написать экзамен по физике», «Если я так буду выглядеть и дальше, на меня никто не посмотрит», «Я ничтожен». А слышали вы что-нибудь об аффирмациях дня? Это короткие высказывания, которые внушают вам веру в себя и настраивают вас на позитивный лад.

Я предлагаю сегодня создать такую программу, которая будет вас каждое утро подбадривать и дарить вам счастье и хорошее настроение на весь день.

Сначала нужно понять, из чего может состоять такая программа. Давайте подумаем. Конечно же, первое, что нам нужно, это создать список аффирмаций. Второе, нужно создать программу, которая будет выбирать для вас случайно одну из аффирмаций и выводить на экран. Вот и вся логика задачи.

Приступим.

Как вы услышали, я использовала слово «случайно» — в Python существует модуль с таким смыслом, он называется *random*. Для чего он нам нужен? Исходя из названия, понятно, что он генерирует что-то случайное. На деле, говоря «случайно», я лукавлю. Роберт Кавью сказал: «Генерация случайных чисел слишком важна, чтобы оставлять ее на волю случая». И был прав. Python выводит «случайные» числа на основе формулы, поэтому правильно говорить, что они «псевдослучайные».

Модуль *random* содержит в себе множество различных методов. Для нашей же задачи нам понадобится один метод, который носит название *choice*. Он позволяет выбрать из заданной последовательности только один элемент. Как его реализовать?

Первое, что сделаем, это импортируем модуль:

```
import random
```

Второе, мы создадим список, из которого программа будет выбирать случайную фразу. Давайте вместе составим содержание нашего списка:

```
affirmation_list = ["Мой возраст идеальный для
 обучения, так как я забочусь о будущем",
 "Я дружу со всеми своими одноклассниками",
 "Мне легко дается учеба, и мне нравится
```

```
 учиться", "Я всегда сосредоточен на учебе", "Мои
 одноклассники — прекрасные люди", "Во мне бушует
 море энергии, и я направляю ее в нужное русло"]
```

Поздравляю, половину программы мы написали.

Теперь мы попросим наше творение выводить на экран любую из фраз. Собственно, применим метод *choice*:

```
print("Аффирмация дня: ",
 random.choice(affirmation_list))
```

Напишем полностью получившуюся программу:

```
import random
affirmation_list = ["Мой возраст идеальный для
 обучения, так как я забочусь о будущем",
 "Я дружу со всеми своими одноклассниками",
 "Мне легко дается учеба, и мне нравится
 учиться", "Я всегда сосредоточен на учебе", "Мои
 одноклассники — прекрасные люди", "Во мне бушует
 море энергии, и я направляю ее в нужное русло"]
print("Аффирмация дня: ",
 random.choice(affirmation_list))
```

Программа написана, но мы можем ее доработать.

Представьте, что вы хотите каждый день на протяжении года получать новую аффирмацию. Что бы вы сделали? Да, возможно, вы бы создали в программе громадный список из аффирмационных выражений. Пришлось бы изрядно полистать, чтобы найти сам код программы.

Или другая ситуация. Вы пишете игру, в которой множество частей. Там описание и ракетки, и мяча, и игрового поля, и ядра программы. Представьте, что все это вы будете хранить в одном файле. Могу с уверенностью сказать, что это жутко неудобно. Начиная с того, что затруднительно читать код, и заканчивая тем, что сложно искать ошибки в каких-то частях.

Для организации пространства в Python существуют подключаемые модули. С некоторыми из них вы уже знакомы. Например, это модули *math*, *time*, *random* и т. д. В этих библиотеках могут содержаться переменные, классы, функции. Если говорить простым языком, это просто файлы, содержащие код. Они нужны для оптимизации программы, для повышения маневренности. Как вы помните, они импортируются с помощью команды *import*.

А теперь вернемся к задаче «Аффирмация дня». Что вы теперь ответите мне на вопрос: «Что сделать, чтобы каждый день на протяжении года мы имели возможность получить разную аффирмацию?»

Именно — нам следует создать отдельный модуль для высказываний. Начнем!

Создание модуля — дело несложное. Достаточно просто сохранить файл с кодом с расширением .py.

Открываем новое окно в среде разработки, пишем там нужный нам список. Пока что нам хватит и тех аффирмаций, которые были записаны. Дома, если хотите, можете написать их сколько угодно.

```
affirmation_list = ["Мой возраст идеальный для
 обучения, так как я забочусь о будущем",
 "Я дружу со всеми своими одноклассниками",
 "Мне легко дается учеба, и мне нравится
 учиться", "Я всегда сосредоточен на учебе", "Мои
 одноклассники — прекрасные люди", "Во мне бушует
 море энергии, и я направляю ее в нужное русло"]
```

Сохраним файл под именем, например, *affirmation.py*. Модуль создан. Дальше берите код, который мы написали выше, и импортируйте в него созданный модуль:

```
import random
import affirmation
print("Аффирмация дня: ", random.
choice(affirmation.affirmation_list))
```

Обратите внимание на то, как именно мы используем список из созданного нами модуля.

В скором времени мы научимся с вами на примере этой же задачи отмечать уже использованные нами элементы списка.

5. Заключение

Описанный в статье подход к разработке сюжетных задач целесообразно использовать как для организации внеурочной деятельности по информатике в VII—IX классах, так и для изучения информатики в основной школе на углубленном уровне [1].

Опыт показывает, что большое количество учащихся, которые на первых занятиях заинтересованы в изучении языка программирования Python, постепенно теряют интерес к нему по мере возрастания сложности изучаемого материала. Задачи, подобные предложенным выше, были использованы автором при проведении занятий во внеурочной деятельности для учащихся VIII классов в Парфеньевской СОШ, что позволило сохранить и развить интерес школьников к изучению программирования.

Список источников

1. *Босова Л. Л.* О новых подходах к изучению школьной информатики в условиях цифровой трансформации общества // Информатика в школе. 2022. № 4. С. 5–14. EDN: DKRLZV. DOI: 10.32517/2221-1993-2022-21-4-5-14

2. *Годфруа Ж.* Что такое психология. М.: Мир, 1992. 496 с.

3. *Головин С. Ю.* Словарь практического психолога. Минск: Харвест, 1998. 800 с.

4. *Гэддис Т.* Начинаем программировать на Python. 4-е изд.: пер. с англ. СПб.: БХВ-Петербург, 2019. 768 с.

5. *Егупова М. В.* Об основных требованиях, предъявляемых к задачам с прикладным содержанием в курсе школьной математики // Наука и школа. 2007. № 3. С. 33–36. EDN: NCRWMD.

6. *Ильин Е. П.* Мотивация и мотивы. СПб.: Питер, 2002. 512 с.

7. *Немов Р. С.* Общая психология. М.: Юрайт, 2016. 262 с.

8. *Патрахина Т. Н., Романчук К. П.* Сущность и содержание понятия «мотивация» в системе управления // Молодой ученый. 2015. № 7. С. 461–464. EDN: TPPMAD.

9. *Платонов К. К.* Занимательная психология. М.: Молодая гвардия, 1986. 224 с.

10. Профессиональный стандарт «Педагог (педагогическая деятельность в сфере начального общего, основного общего, среднего общего образования) (учитель)» (проект подготовлен Минтрудом России 31.01.2022). https://www.garant.ru/products/ipo/prime/doc/56809182/

11. Федеральный закон от 29 декабря 2012 года № 273-ФЗ (ред. от 29.07.2017) «Об образовании в Российской Федерации». http://www.consultant.ru/document/cons_doc_law_140174/

12. *Фридман Л. М.* Сюжетные задачи по математике. История, теория, методика: учебное пособие для учителей и студентов педвузов и колледжей. М.: Школьная пресса, 2002. 204 с.