Задание 11

https://yandex.ru/tutor/subject/problem/?problem_id=T4822

В качестве задания приводится рекурсивный алгоритм, записанный на нескольких языках программирования.

Например, рассмотрим следующий вариант:

Pascal

```
procedure F(n: integer);
begin
  if n < 8 then begin
    F(n + 3);
    write(n);
    F(2 * n)
  end
end;
```

Python

```
def F(n):
  if n < 8:
    F(n + 3)
    print(n)
    F(2 * n)
```

Запишите подряд без пробелов и разделителей все числа, которые будут показаны на экране при выполнении вызова функции F(1). Числа должны быть записаны в том же порядке, в котором они выводятся на экран.

Решение

Данный тип задач можно отнести к достаточно простым.

Чтобы успешно решать такого рода задачи вы должны понимать, как оформляются и работают функции, понимать логику операторов ветвления и рекурсивных функций. Рекомендуется решить 5-7 задач такого рода, представленных на одном из алгоритмических языков программирования, для получения твёрдых навыков решения задач «на рекурсию». Более того, решать рекурсивные задачи можно не умея программировать, достаточно лишь понимать синтаксис описания рекурсивной функции.

Приступаем к разбору.

Разбор будет состоять из двух частей:

1) первая – вводная – прояснит вопрос с функциями и рекурсивными функциями,

2) вторая – собственно пошаговый разбор работы рекурсивной функции этой задачи.

Часть 1.

Прежде всего, определимся с понятием функция.

Функция – это часть программного кода (она вынесена отдельно от основной программы), к которой можно обратиться по имени. Функция может выполнится одни раз, много раз или не выполнится ни разу – это зависит от того обратились к ней из основного текста программы или нет. Более того, функция может вызывать сама себя, тогда она называется рекурсивной. Конечно, чтобы она начала работать и вызывать сама себя нужно, чтобы хотя бы один раз к ней было обращение из основного текста программы. Функция может получать один параметр, несколько параметров или ни одного параметра. Параметры передаются через переменные, которые могут быть записаны после названия функции в круглых скобках, например, **`F(1)`** – это функция F, в которую передают один параметр, равный **`1`**. Чтобы эта функция заработала, она должна была быть описана (продекларирована) где-то ранее, например, так это может быть сделано на Питоне:

```
def F(x):
    print(x)
```

В данном случае, продекларирована (def от слова define – определить) функция **`F`**, которая может принять в переменную **`x`** некоторое значение и вывести его на экран. Если из основного текста программы обратиться к этой функции несколько раз, то она и сработает несколько раз, например, так:

```
x = int(input())
F(x)
x = x + 1
F(x)
```

В этой программе предполагается, что пользователь введёт какое-то число, которое будет выведено на экран функцией **F**, потом увеличено на единицу и снова выведено на экран вторым вызовом функции **F**.

Мы рассмотрели пример программы с обычной функцией, а теперь рассмотрим оформление программы с рекурсивной функцией:

```
def F(x):
    if x > 5:
        print(x)
    else:
        F(x+1)

x = int(input())
F(x)
```

В этой программе вы видите в самом начале код функции **F**:

```
def F(x):
    if x > 5:
        print(x)
    else:
        F(x+1)
```

и далее код основной программы, состоящий из двух строчек:

```
x = int(input())
F(x)
```

предусматривающей ввод целого числа и последующий запуск функции **F** с введённым числом на исполнение.

Если пользователь введёт число 1, то во второй строчке на исполнение будет вызвана функция **F** с параметром, равным 1 – **F(1)** – именно так примерно и оформляется вопрос в задании 11, то есть вот вам дана декларация функции и вот вам её вызов – отвечайте, что будет выведено на экран.

Приведённая выше рекурсивная функция достаточно проста, чтобы можно было бы с неё начать изучение работы рекурсивных функций. Исследуем, что именно делает эта функция – вот тело функции:

```
    if x > 5:
        print(x)
    else:
        F(x+1)
```

В функции сначала проверяется значение переменной **x**, если оно больше **5**, то печатаем значение на экран (и всё, больше ничего не делаем), а иначе – вызываем на исполнение функцию **F** с параметром **x**, увеличенным на единицу.

Таким образом, если изначально запустить функцию F(1), то так как в переменную x будет передано значение 1 (это не больше 5), то включается путь ИНАЧЕ и функция просто вызывается со значением параметра 2. Для

F(2) аналогичная ситуация и происходит рекурсивный вызов F(3), далее F(4), потом F(5), и, наконец, при значении параметра 6 происходит печать значения 6 и всё, работа функции завершается – за все эти вызовы только один раз был осуществлён вывод на экран.

Теперь немного изменим функцию:

```
def F(x):
    if x > 0:
        print(x)
        F(x-1)
    else:
        print(x)

x = int(input())
F(x)
```

Пусть пользователь введёт число 3, тогда будет вызов функции F(3). Это приведёт к следующей последовательности вызовов:

Первая итерация: в функцию попадает значение 3, так как 3>0, то производится вывод **3** на экран и рекурсивный вызов функции со значением, уменьшенным на 1, то есть F(2).

Вторая итерация: в функцию попадает значение 2, так как 2>0, то производится вывод **2** на экран и рекурсивный вызов функции со значением, уменьшенным на 1, то есть F(1).

Третья итерация: в функцию попадает значение 1, так как 1>0, то производится вывод **1** на экран и рекурсивный вызов функции со значением, уменьшенным на 1, то есть F(0).

Заключительная итерация: в функцию попадает значение 0, так как 0 не больше 0, то осуществляется переход к ветви ИНАЧЕ, где производится вывод **0** на экран и всё – работа рекурсивной функции на этом завершается.

Если бы именно эта задача была поставлена в задании №11, то по условию мы должны были бы записать последовательно все ответы в одну строчку без пробелов, то есть так: **3210** – это и был бы правильный ответ.

Однако, в задании №11, как правило, дают более сложную рекурсивную функцию, в которой тело функции содержит больше действий.

Часть 2.

Переходим к разбору функции из задания:

```
def F(n):
  if n < 8:
    F(n + 3)
    print(n)
    F(2 * n)
```

По требованиям задачи производится вызов функции **`F(1)`**.

Основная особенность этой функции состоит в том, что в теле функции **три команды**, две из которых это рекурсивный вызов самой же функции с изменёнными значениями, причём каждый раз нужно проверять истинность некоторого условия. Тут нужно учитывать (это очень важно), что команды в теле функции исполняются исключительно последовательно, то есть переход к следующей команде происходит только после полного окончания выполнения предыдущей. Например, обратите внимание, что команда print(n) выполняется после F(n+3), а ведь F(n+3) это рекурсивная функция, которая сама будет ещё что-то вызвать рекурсивно…

Все эти рекурсивные вызовы затруднительно удержать «в голове», поэтому имеет смысл изобразить последовательность действий в виде графа вызовов функций.

Для начала определимся с обозначением **одного вызова функции** – пусть это внешне будет выглядеть как одномерная таблица, например, вызов функции **`F(1)`** содержит в теле три команды:

`F(4)`	`1`	`F(2)`

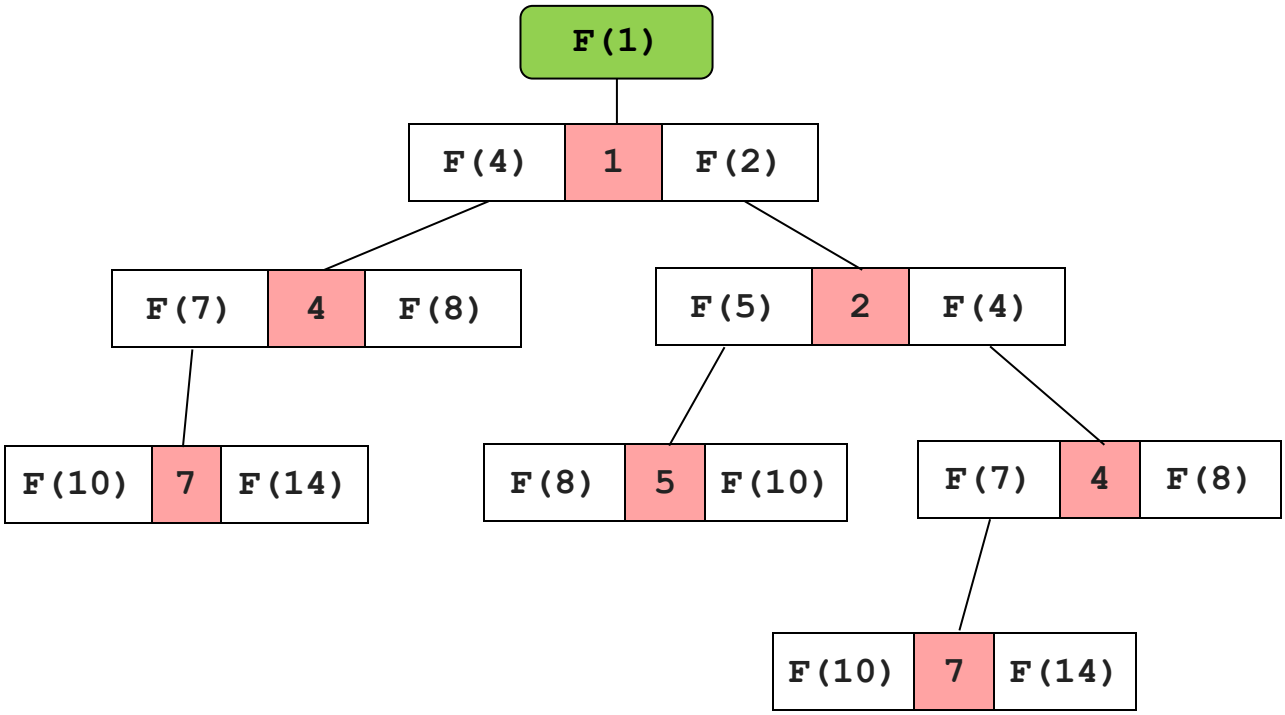
В первой ячейке написано F(4), так как первая команда в теле заданной рекурсивной функции определена как F(n+3), а в данный момент в качестве параметра функции n было передано значение 1 (1+3=4).

Во второй ячейке написано число 1, но, на самом деле, должно быть написано print(1), однако для краткости оставим только число. Далее будем поступать так же – в центральной ячейке будем писать только число.

В третьей ячейке написано F(2), так как третья команда в теле заданной рекурсивной функции F(2*n) – тут n=1 умножаем на 2 и получаем 2.

Уточню ещё раз, что при первом вызове в функцию было передано число 1 и оно пока не менялось, то есть и при выполнении первой и второй и третьей команды в рамках одной итерации (текущего вызова) переменная n хранит одно и тоже значение. Но, когда мы обращаемся к рекурсивному вызову, то передаём в качестве параметра **изменённое значение**: для данной задачи – это либо **`n+3`**, либо **`2*n`**.

Учитывая, что тело функции **будет исполняться только при `n<8`**, нарисуем полный граф рекурсивных вызовов:



На данном графе красным цветом залиты команды print с обозначением соответствующих значений для вывода, а квадратиками с функциями без исходящих стрелок обозначены те функции, тело которых не будет исполняться (не соответствует условию n<8).

Оценим последовательность исполнения (смотрите по графу).

На верхнем (первом) уровне иерархии графа обозначен вызов F(1), который приводит к последовательному исполнению F(4), print(1), F(2), но до команды print(1) мы дойдём только после полного исполнения F(4) со всеми её рекурсивными вызовами. Поэтому сначала смотрим, что нам предлагает вызов F(4), а там, в свою очередь, такая последовательность: F(7), print(4), F(8), но и тут до print(8) мы дойдём только после исполнения F(7).

Теперь смотрим на **вызов F(7)** – там последовательность F(10), print(7), F(14). Тут F(10) не будет исполняться, так как не соответствует условию n<8, далее **выполняется print(7)**, а F(14) тоже не будет исполняться, так как не соответствует условию n<8. Таким образом, первое выведенное на экран число **7** и на этом **завершается выполнение вызова F(7)**.

После завершения вызова F(7) (смотрим по графу) у нас **выполняется print(4)**, после которого следует вызов F(8), который ни к чему не приводит, так как не соответствует условию n<8. На этом **завершается вызов F(4)**.

После завершения вызова F(4) (смотрим по графу) **выполняется print(1)**, после которого следует вызов F(2).

К этому моменту уже были напечатаны числа, 7, 4 и 1. Далее все рекурсивные вызовы следует рассматривать именно в последовательности исполнения вызовов по графу и команд в теле функции слева-направо.

Так как в качестве ответа требуется записать последовательность выводимых чисел без пробелов в виде одной строки, то в этой задаче правильная запись ответа будет такой: **`7415274`**.