# Упражнения по программированию главы 12

```
# coding: utf-8
```

## Упражнение по программированию 12.1. Рекурсивная печать

```
def main():
 # Локальная переменная.
 number = 0

 # Получить от пользователя число.
 number = int(input('Сколько показать чисел? '))

 # Показать числа.
 print_num(number)

# Функция print_num является рекурсивной функцией, которая
# принимает целочисленный аргумент n и печатает числа от 1 до n.
def print_num(n):
    if n > 1:
        print_num(n - 1)
    print (n, sep=' ')

# Вызвать главную функцию.
main()
```

## Упражнение по программированию 12.2. Рекурсивное умножение

```
def main():
    # Локальные переменные.
    num1 = 0
    num2 = 0

    # Получить от пользователя первое неотрицательное целое число.
    while num1 <= 0:
        num1 = int(input('Введите первое число: '))

    # Получить от пользователя второе неотрицательное целое число.
    while num2 <= 0:
        num2 = int(input('Введите второе число: '))

    # Вызвать функцию multipy и показать произведение.
    print(num1, 'умножить на', num2, 'равняется', multiply(num1, num2))

# Функция multiply является рекурсивной функцией, которая
# принимает два аргумента в параметры x и y, а затем возвращает
# значение x, умноженное на y.
# Эта функция исходит из того, что x и y всегда содержат неотрицательные целые числа.
def multiply(x, y):
    if x == 0 or y == 0:
```

```
        return 0
    else:
        return x + multiply(x, y - 1)

# Вызвать главную функцию.
main()
```

## Упражнение по программированию 12.3. Рекурсивные строки

```
def main():
   # Локальная переменная.
   number = 0

   # Получить от пользователя количество строк.
   number = int(input('Сколько строк показать? '))

   # Показать строки.
   print_lines(number)

# Функция print_lines является рекурсивной функцией, которая
# принимает целочисленный аргумент n. Функция показывает
# на экране n строк, состоящих из звездочек. При этом первая
# строка показывает 1 звездочку, вторая строка - 2 звездочки,
# вплоть до n-й с n звездочками.
def print_lines(n):
    if n > 1:
        print_lines(n - 1)
    for i in range(n):
        print('*', end='')
    print()

# Вызвать главную функцию.
main()
```

## Упражнение по программированию 12.4. Максимальное значение в списке

```
def main():
    # Инициализировать список значений от 1 до 10.
    number_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    # Показать список.
    print('Список чисел:\n', number_list, sep='')

    # Вызвать функцию find_largest и показать
    # максимальное число в спике.
    print('Максимальное число в списке: ', find_largest(number_list))

# Функция find_largest принимает список в качестве аргумента и
# возвращает максимальное значение в списке.
# Для нахождения максимального значения эта функция использует рекурсию.
```

```
def find_largest(numlist):
    n  = len(numlist)
    if n == 1:
        return numlist[0]
    else:
        temp = find_largest(numlist[0:n-1])
        if numlist[n-1] > temp:
            return numlist[n-1]
        else:
            return temp

# Вызвать главную функцию.
main()
```

## Упражнение по программированию 12.5. Рекурсивная сумма списка

```
def main():
    # Инициализировать список значений от 1 до 10.
    number_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    # Показать список.
    print('Список чисел:\n', number_list, sep='')

    # Вызвать функцию sum_list и показать сумму всех
    # чисел в списке.
    print('Сумма всех чисел в списке:', \
          sum_list(number_list))

# Функция sum_list принимает список чисел в качестве аргументов.
# Эта функция рекурсивно вычисляет сумму всех чисел в списке и
# возвращает значение.
def sum_list(numlist):
    n  = len(numlist)
    if len(numlist) == 1:
        return numlist[0]
    else:
        return numlist[n-1] + sum_list(numlist[0:n-1])

# Вызвать главную функцию.
main()
```

## Упражнение по программированию 12.6. Сумма чисел

```
def main():
    # Локальная переменная.
    num = 0

    # Получить от пользователя число.
    while num <= 0:
        num = int(input('Введите целое положительное число: '))
```

```
    # Вызвать функцию sum_num и показать сумму.
    print('Сумма от 1 до', num, 'равняется', format(sum_nums(num), ','))

# Функция sum_nums принимает целочисленный аргумент и возвращает
# сумму всех целых чисел, начиная с 1 до числа, переданного в
# качестве аргумента.
def sum_nums(n):
    if n == 0:
        return n
    else:
        return n + sum_nums(n-1)

# Вызвать главную функцию.
main()
```

## Упражнение по программированию 12.7. Рекурсивный метод возведения в степень

```
# Глобальные константы для минимального и максимального
# показателя степени
MIN = 1
MAX = 100

def main():
    # Локальные переменные.
    num = 0.0
    exp = 0

    # Получить от пользователя число.
    num = float(input('Ввести число: '))

    # Получить от пользователя показатель степени.
    while exp < MIN or exp > MAX:
        exp = int(input('Введите положительное целое число между ' \
                        + str(MIN) + ' и ' + str(MAX) + ': '))

    # Вызвать функцию power и показать результат.
    print(f'{num:.2f}, возведенное в степень {exp}, равняется {power(num, exp):.2f}')

# Функция power использует рекурсию для возведения числа в степень.
# Эта функция принимает два аргумента: число, возводимое в степень, и
# показатель степени.
# Эта функция исходит из того, что показатель степени является
# неотрицательным целым числом.
def power(x, y):
    if y == 0:
        return 1
    else:
        return x * power(x, y - 1)
```

```
# Вызвать главную функцию.
main()
```

## Упражнение по программированию 12.8. Функция Аккерманна

```
def main():
    # Тестовое значение 1
    num1 = ackermann(0, 3)
    print(num1)

    # Тестовое значение 2
    num2 = ackermann(2, 0)
    print(num2)

    # Тестовое значение 3
    num3 = ackermann(2, 3)
    print(num3)

# Функция Аккерманна является рекурсивным математическим алгоритмом,
# который используется для проверки, насколько успешно система
# оптимизирует свою производительность в случае рекурсии.
def ackermann(m,n):
    if m == 0:
        return n + 1
    elif n == 0:
        return ackermann(m - 1, 1)
    else:
        return ackermann(m - 1, ackermann(m, n - 1))

# Вызвать главную функцию.
main()
```