работа на уроке (13.03)

1. `def sum_sub()` `% 2`

   Напишите рекурсивную функцию `sum_sub()`, которая будет принимать список целых чисел (`numbers`). Эта функция будет суммировать все нечётные числа и вычитать все чётные числа. В конце она будет возвращать получившееся значение.

Вызов	Возвращение
`>>> sum_sub([1,2,3,4])`	`>>> -2`
`>>> sum_sub([1,2,3,4,5])`	`>>> 3`

2. `def friend_of_friend()` `.reverse()` `% 2`

   У вас есть словарь `friends`, который представляет из себя сеть друзей. Например, такой:

```
friends = {
          1: [2, 3], # 1-й дружит с 2-м и 3-м
          2: [1], # 2-й дружит с 1-м
          3: [1, 4], # 3-й дружит с 1-м и 4-м
          4: [3], # и т. д.
          5: [6],
          6: [5, 7],
          7: [6]
          }
```

   Напишите рекурсивную функцию `friend_of_friend()`, которая будет принимать `id` человека и словарь `friends` и находить всех его друзей, друзей их друзей и т. д.…

Вызов	Возвращение
`>>> friends(4)`	`>>>[1,2,3,4]`
`>>> friends(5)`	`>>> [5,6,7]`

4. `def choco()`

   `math.floor() / int()`

На фабрике Вилли Вонки каждый ребенок покупает шоколад на все деньги, что у него есть. Кроме того, действует еще одно правило: ты можешь обменять некоторое количество упаковок от шоколадок на новую шоколадку. Однако Вилли очень непостоянен, поэтому как цена на шоколад, так и “курс обмена” меняется каждый день. Помогите Чарли понять, сколько шоколадок он сможет съесть на имеющиеся у него деньги.
Рекурсивная функция `choco()` принимает на вход три числа: `money` — количество денег у Чарли, `price` — цену одной шоколадки и `wrap` — количество упаковок, которые нужно сдать, чтобы получить еще одну шоколадку. Возвращает она максимальное число шоколадок, съеденных Чарли.

Вызов	**Возвращение**
`>>> choco(15, 1, 3)`	`>>> 22`

5. `def find_videoid()`
   `type()`

   Вам дан неупорядоченный словарь, содержащий в себе несколько вложенных структур данных. Некоторые из них (независимо от вложенности) содержат ключ *VideoID*.

```
data = {"type": "video",
        "videoID": "vid001",
        "links": [
                    {"type":"video",
                    "videoID":"vid002",
                    "links":[]
                    },
                    {"type":"video",
                    "videoID":"vid003",
                    "links": [
                                {"type": "video",
                                "videoID":"vid004"
                                },
                                {"type": "video",
                                "videoID":"vid005"
                                },
                              ]
                    },
                    {"type":"video",
                    "videoID":"vid006"
                    },
                    {"type":"video",
                    "videoID":"vid007",
                    "links": [
                                {"type":"video",
```

```
                                "videoID":"vid008",
                                "links": [
                                            {"type":"video",
                                            "videoID":"vid009",
                                            "links": [
                                                        {"type":"video",
                                                        "videoID":"vid010"
                                                        }
                                                    ]
                                            }
                                        ]
                                }
                            ]
                    },
                ]
        }
```

Напишите рекурсивную функцию `find_videoid()`, целью которой будет извлечение всех значений с ключом *VideoID* из переданного в функцию словаря `data`. Если таковых нет, верните пустой список!

```
output = [{'videoID': 'vid001'}, {'videoID': 'vid002'},
          {'videoID': 'vid003'}, {'videoID': 'vid004'},
          {'videoID': 'vid005'}, {'videoID': 'vid006'},
          {'videoID': 'vid007'}, {'videoID': 'vid008'},
          {'videoID': 'vid009'}, {'videoID': 'vid010'}
          ]
```

6. `def maze_runner()`

Вам дам список строк, который представляет из себя лабиринт.

```
maze = ['S----',
        '##---',
        '---##',
        '----X'
        ]
```

`S` — стартовая позиция (откуда игрок начинает);
`X` — финиш (цель прохождения лабиринта);
`-` — свободный путь, по которому может осуществлять движение игрок;
`#` — стена, через которую игрок не может пройти.

Правила:

- Игрок может двигаться только на 1 шаг за один раз;
- Игрок может двигаться только в 4 направлениях: *вверх*, *вниз*, *влево*, *вправо*.
- Лабиринт проходим, если игрок может пройти до `X` из `S`.

Напишите рекурсивную функцию `maze_runner()`, которая будет принимать лабиринт `maze` и возвращать `True`, если он проходим, и `False` в любом другом случае.

7. `def geometric_progression()`

   Напишите рекурсивную функцию `geometric_progression()` для вычисления суммы `n` первых членов геометрической прогрессии:

$$\sum_{n=1}^{\infty} \frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}$$

Вызов	Возвращение
`>>> geometric_progression(9)`	`>>>1.99609375`

5. `def power_of_two()`

   На вход рекурсивной функции `power_of_two()` подается число `number`. Проверьте, является ли `number` степенью двойки и верните `True` или `False` соответственно.

Вызов	Возвращение
`>>> power_of_two(2048)`	`>>> True`
`>>> power_of_two(192)`	`>>> False`

P. S. Напишите аналогичную функцию `power_of_number()`, которая принимала бы на вход два числа: `number_1` — степень и `number_2` — число для проверки, — и вычисляла бы, является ли `number_2` степенью `number_1`.