Практические занятия №1-2

1. Первое знакомство с языком

1.1. (0 баллов)

В реализации Питона спрятано несколько «пасхальных яиц». С одним из них мы уже сталкивались, это `import this`. Есть «пасхалка» связанная с гравитацией - `import antigravity`. Еще одна - `import __hello__` или `import __phello__`, для тех, кто ленится написать даже самую первую программу, а также `from __future__ import braces` для любителей языков программирования с непохожим на Питон синтаксисом. А еще есть сюрприз для тех, кто хотел бы сравнивать объекты, как в Паскале - `from __future__ import barry_as_FLUFL`.

1.2. (0 баллов)

Придумайте несколько способов записи числа 42. Необходимо использовать только литеральную запись, без арифметики и функций. В любом варианте должно соблюдаться == 42.

1.3. (0 баллов)

С приведенным ниже циклом что-то не так. Как это исправить?

Что на самом деле представляет собой 0.1 внутри интерпретатора? Можно ли увидеть его настоящее значение? Поэкспериментируйте: https://float.exposed/0x44bf9400

In [ ]:

```
a = 10
while a != 0:
    a -= 0.1
```

1.4. (0 баллов)

А вот совсем уже безобидный код. Циклов нет, но программа зависает. Почему?

In [ ]:

```
z = 1
z <<= 40
2 ** z
```

1.5. (0 баллов)

Да-да, и этот код зацикливается! А тут-то что не так?

In [ ]:

```
i = 0
while i < 10:
    print(i)
    ++i
```

1.6. (0 баллов)

Что за странное выражение и странный результат?

In [ ]:

```
(True * 2 + False) * -True
```

Out[ ]:

```
-2
```

1.7. (0 баллов)

В Питоне можно использовать цепочки операций сравнения. Рассмотрите следующие примеры и попробуйте объяснить код:

In [14]:

```
x = 5
1 < x < 10
```

Out[14]:

```
True
```

In [15]:

```
x = 5
1 < (x < 10)
```

Out[15]:

```
False
```

In [16]:

```
x = 5
1 < x < (x < 10)
```

Out[16]:

```
False
```

2. Сообщения об ошибках

2.1. (0 баллов)

К сообщениям об ошибках Питона нужно привыкать, в них нет ничего страшного. Давайте специально напишем некорректный код для того, чтобы получить каждое из указанных ниже сообщений об ошибках.

1. `SyntaxError: invalid syntax`
2. `SyntaxError: cannot assign to literal`
3. `NameError: name ... is not defined`
4. `SyntaxError: unterminated string literal (с v3.10)` или `SyntaxError: EOL while scanning string literal (до v3.10)`
5. `TypeError: unsupported operand type(s) for ...`
6. `IndentationError: expected an indented block`
7. `IndentationError: unindent does not match any outer indentation level`
8. `ValueError: math domain error`
9. `OverflowError: math range error`

3. Арифметика и логика

Мир микропроцессоров не ограничивается только большими чипами для настольных применений. Маломощные микроконтроллеры могут выступать в роли умных датчиков в задачах, связанных с Интернетом вещей. Слабенькие 8-битные микропроцессоры являлись «сердцем» многих классических игровых приставок. Типичный 8-битный процессор не имеет аппаратной поддержки умножения. Как же выкручиваются программисты в этой ситуации?

Представим, что в Питоне тоже отсутствует операция умножения. Ее можно заменить сложением. Если мы хотим умножить какое-то число x на 12, то нам понадобится 11 сложений, правильно? Это довольно много, но, оказывается, можно обойтись и меньшим числом сложений.

Из арифметических операций разрешается использовать только явно указанные и в указанном количестве. Входным аргументом является переменная x. Унарный минус использовать нельзя, разрешается только бинарный. Тело программы должно состоять из линейной последовательности присваиваний. Оформите линейный код решения в виде функции.

3.1. (0.1 балла)

Умножение на 12. Используйте 4 сложения, одну переменную.

3.2. (0.1 балла)

Умножение на 16. Используйте 4 сложения, одну переменную.

3.3. (0.1 балла)

Умножение на 15. Используйте 3 сложения, 2 вычитания и две переменные.

3.4. (0.2 балла)

Вычислить значение итерационной функции:

$$f(b,n,a) = \sum_{j=1}^{n}\sum_{c=1}^{b}\left((34j+41)^4 - 93\left(c+79+c^3\right)^5\right) - \prod_{k=1}^{a}\sum_{c=1}^{b}\left(22(c-8)^5 - k^4\right), \qquad (1)$$

при $b = 2, n = 2, a = 6$.

3.5. (0.2 балла)

Вычислить значение кусочной функции:

$$f(x) = \begin{cases} x^5, & x < 13 \\ x^7 - 1 - \frac{(\lfloor x \rfloor)^3}{54}, & 13 \le x < 87 \\ (\lceil x \rceil)^3, & x \ge 87 \end{cases} \qquad (2)$$

при $x = 14$. Скобки `[]` с отсутствующей верхней или нижней горизонтальной чертой обозначают операцию взятия целой части числа.

3.6. (0.2 балла)

Вычислить значение рекуррентной функции:

$$f(n) = \sin(f(n-1)) - \frac{1}{16}f(n-1)^3 \qquad (3)$$

при $f(0) = 3$ и $n = 8$.

Существует старинный метод умножения по методу русского крестьянина. Разобрать его проще на примере.

Предположим, мы хотим перемножить 10 и 15:

x	y
10	15
5	30
2	60
1	120

В первом столбце последовательно записывают результаты деления на 2 с отбрасыванием остатка. Во втором столбце находятся результаты умножения на 2. Отмечаем нечетные числа в первом столбце. Складываем те числа в правом столбце, которые стоят напротив отмеченных ранее чисел. То есть 30 + 120 = 150. В нашем случае мы не учитывали исходные числа при сложении, но учитывать их, вообще говоря, может быть нужно.

В этом алгоритме используются лишь простейшие операции: умножение на 2, целочисленное деление на 2, проверка на нечетность и сложение. Эти операции соответствуют тем элементарным действиям, которые эффективно выполняет любой процессор. Сам же алгоритм, несмотря на кажущуюся необычность, является вариантом умножения в столбик при использовании двоичного представления чисел:

```
      1111
      1010 *
      ----
      0000
     1111
    0000
   1111
```

3.7. (0.3 балла)

Реализуйте функцию fast_mul в соответствии с алгоритмом двоичного умножения в столбик (без рекурсии и использования сторонних функций, таких как bin() !). При желании добавьте автоматическое тестирование: https://clck.ru/38jeQX

3.8. (0.3 балла)

Реализуйте аналогичную функцию fast_pow для возведения в степень. Решение необходимо получить только с помощью небольших модификаций предыдущего решения.

Иногда возникает необходимость в создании программы, которая в качестве результата выдаст другую программу. В этом нет ничего необычного, так устроен, к примеру, компилятор. Но если задача стоит узкоспециальная, то речь идет о создании генератора программ по заданным параметрам.

3.9. (0.3 балла)

Реализуйте генератор программ fast_mul_gen(y) для задач 3.1-3.3. Ваша функция должна выдать текст функции f(x) (умножение на ранее заданный y), тело которой состоит из некоторого числа присваиваний. Для вывода функции используйте print. Добавьте автоматическое тестирование.

4. Пиксельные шейдеры *

Шейдеры представляют собой небольшие программы, обычно предназначенные для исполнения на графической карте. Шейдеры могут работать параллельно и не запоминают свое состояние. Это просто функции, переводящие координаты экрана в цвет. Шейдеры широко используются для создания специальных эффектов, а также в играх. Обычно шейдеры программируют на C-подобных языках. Попробуем имитировать работу шейдеров прямо в Питоне!

Итак, вся работа должна производится в теле функции func(x, y). Координаты заданы в диапазоне [0, 1). Функция возвращает три цветовых компонента, каждый из которых также находится в диапазоне [0, 1]. При решении задач старайтесь не использовать ветвлений и, тем более, циклов. Не забывайте, что шейдеры не имеют доступа к глобальным данным и поэтому даже модуль random использовать нельзя.

Для решения задач понадобится приведенная ниже заготовка программы.

Дополнительная информация

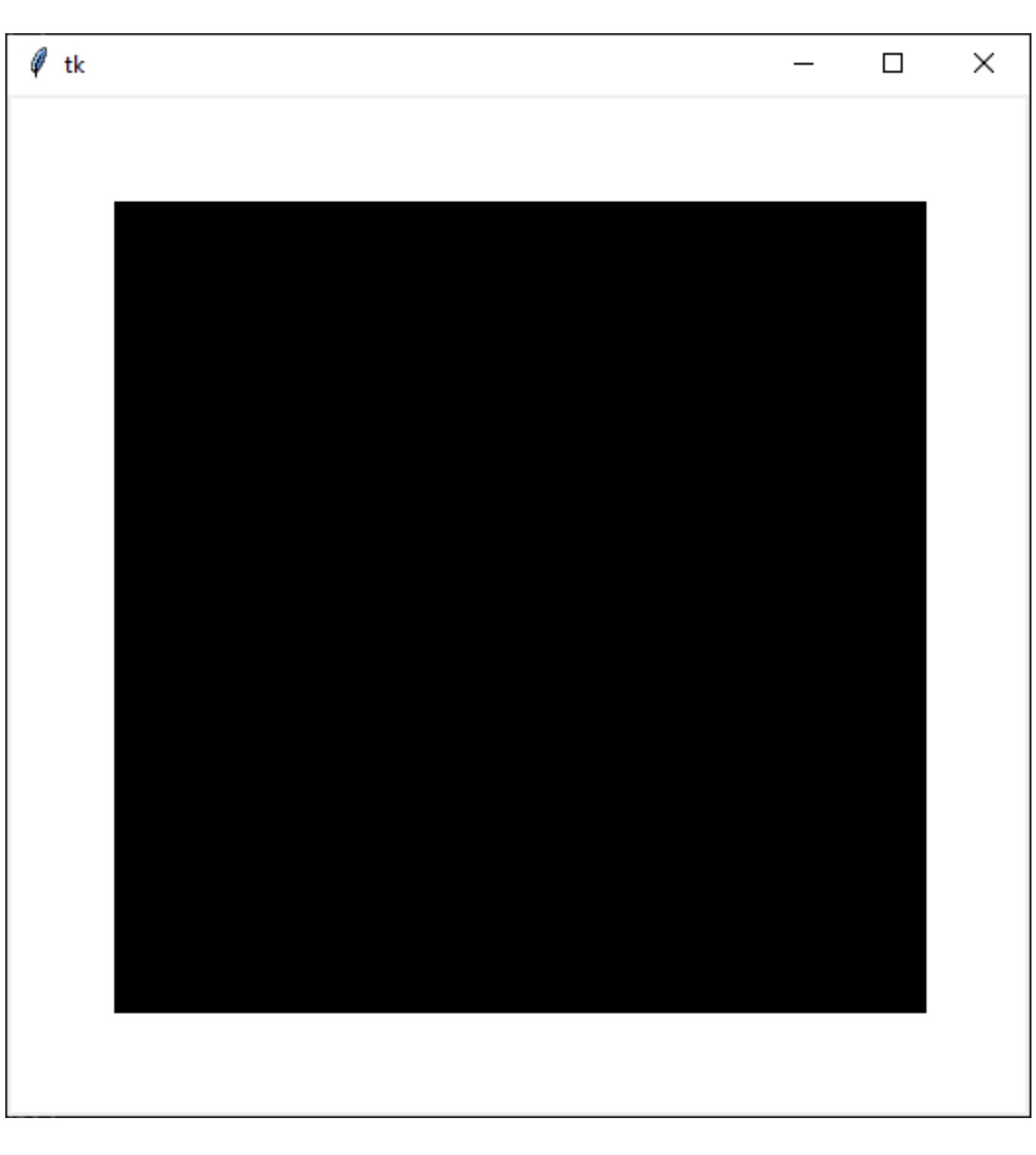
1. Книга The Book of Shaders: https://thebookofshaders.com/.
2. Сайт известного специалиста в области процедурной графики Inigo Quilez: https://iquilezles.org/articles/.
3. О Value Noise: https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/procedural-patterns-noise-part-1/introduction.html.

In [ ]:

```
import math
import tkinter as tk


def pyshader(func, w, h):
    scr = bytearray((0, 0, 0) * w * h)
    for y in range(h):
        for x in range(w):
            p = (w * y + x) * 3
            scr[p:p + 3] = [max(min(int(c * 255), 255), 0)
                            for c in func(x / w, y / h)]
    return bytes('P6\n%d %d\n255\n' % (w, h), 'ascii') + scr


# Ваш код здесь:
def func(x, y):
    return x, y, 0


label = tk.Label()
img = tk.PhotoImage(data=pyshader(func, 256, 256)).zoom(2, 2)
label.pack()
label.config(image=img)
tk.mainloop()
```

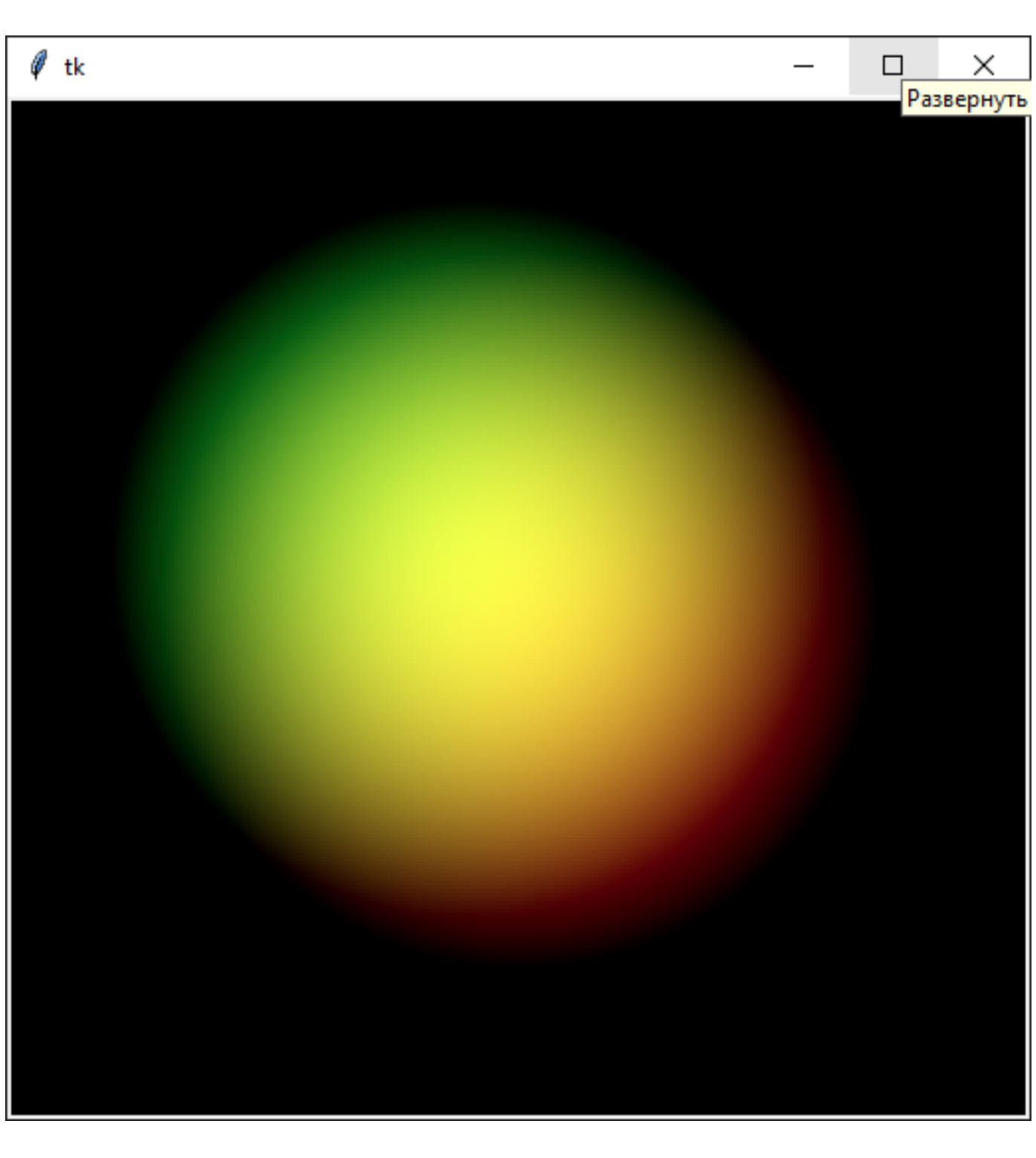
4.1. (0.1 балла)

Изобразите свою версию знаменитого «Черного квадрата».



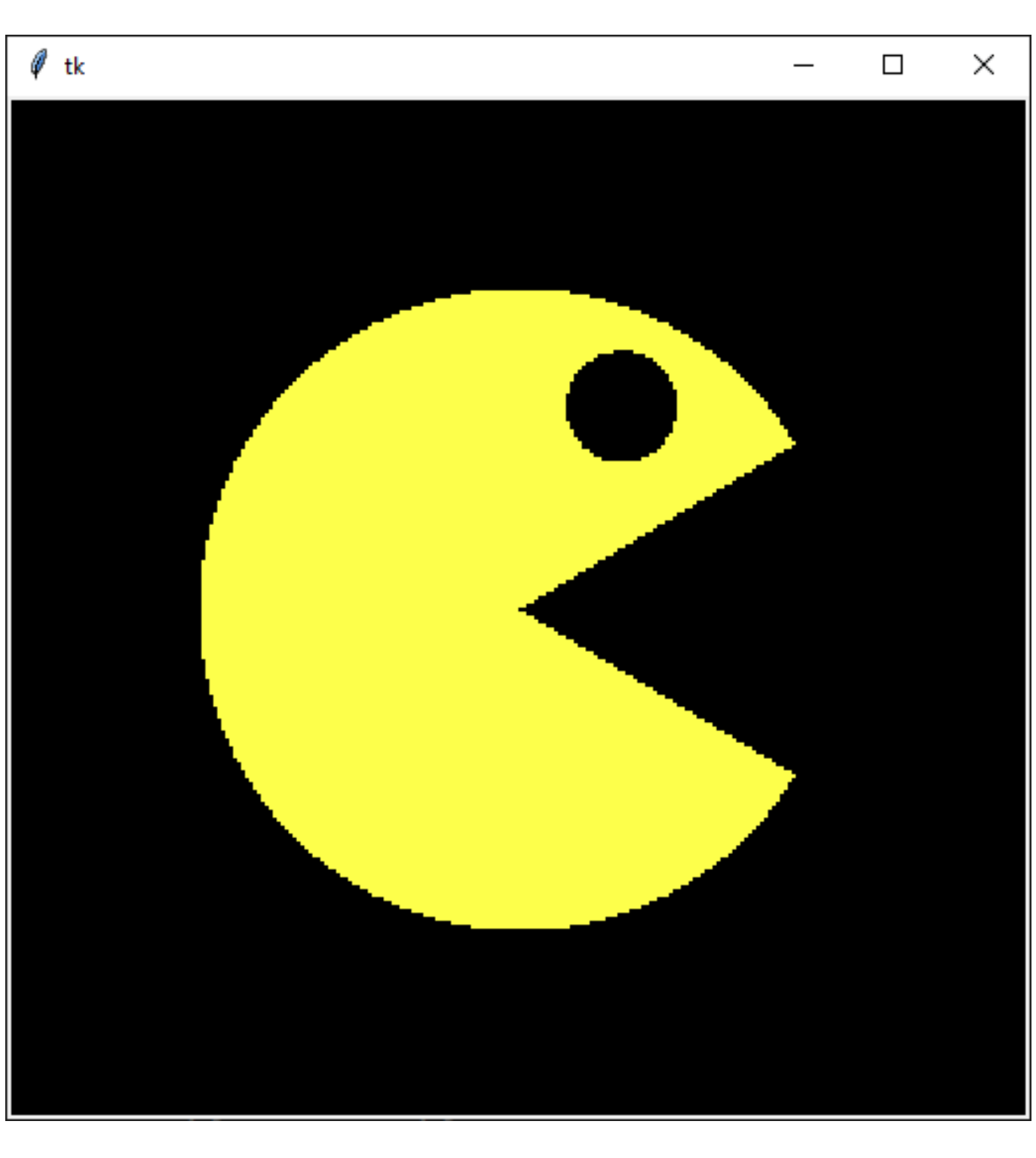
4.2. (0.3 балла)

Изобразите шар, показанный на примере ниже.



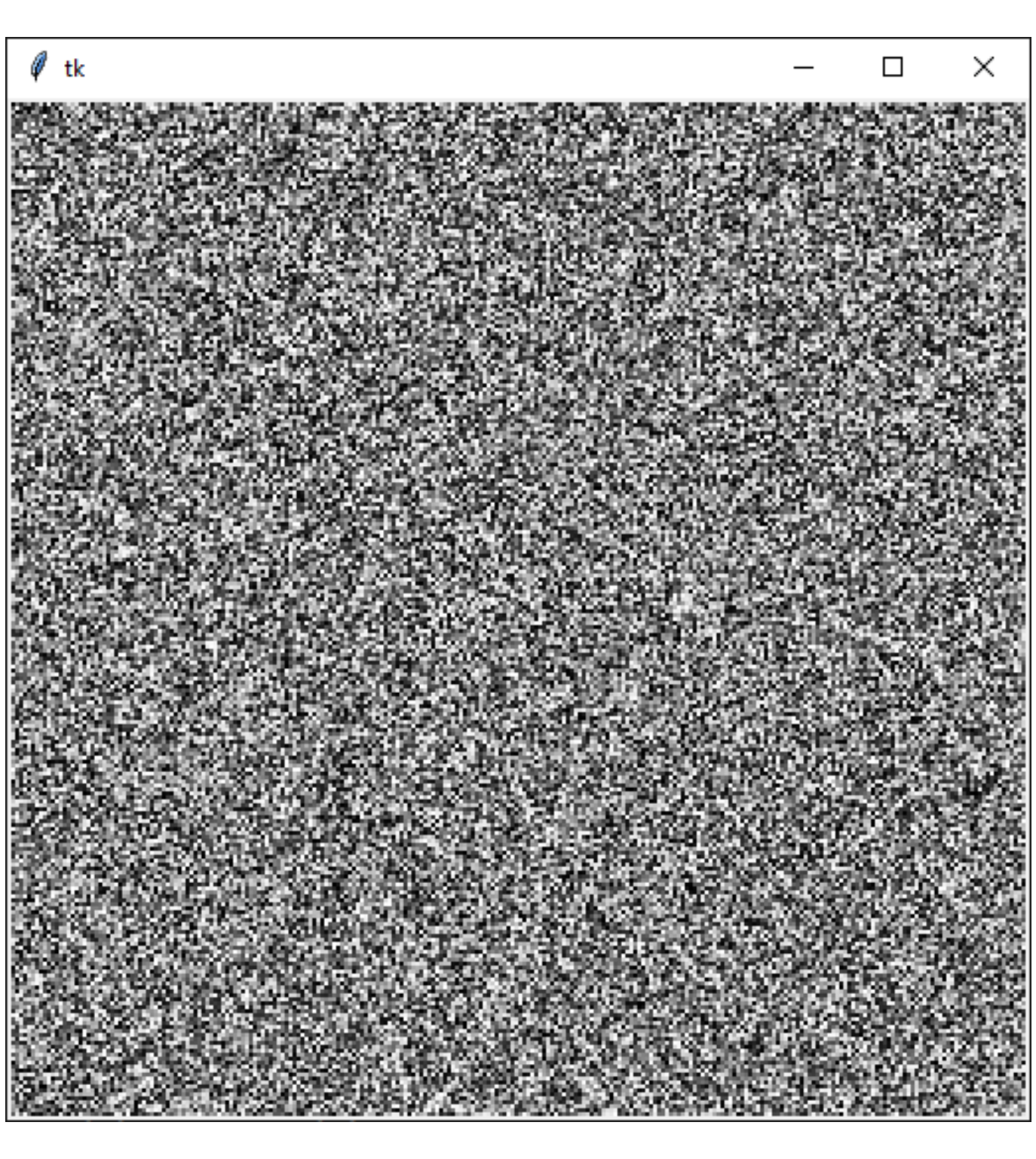
4.3. (0.4 балла)

Изобразите знаменитого персонажа компьютерной игры.



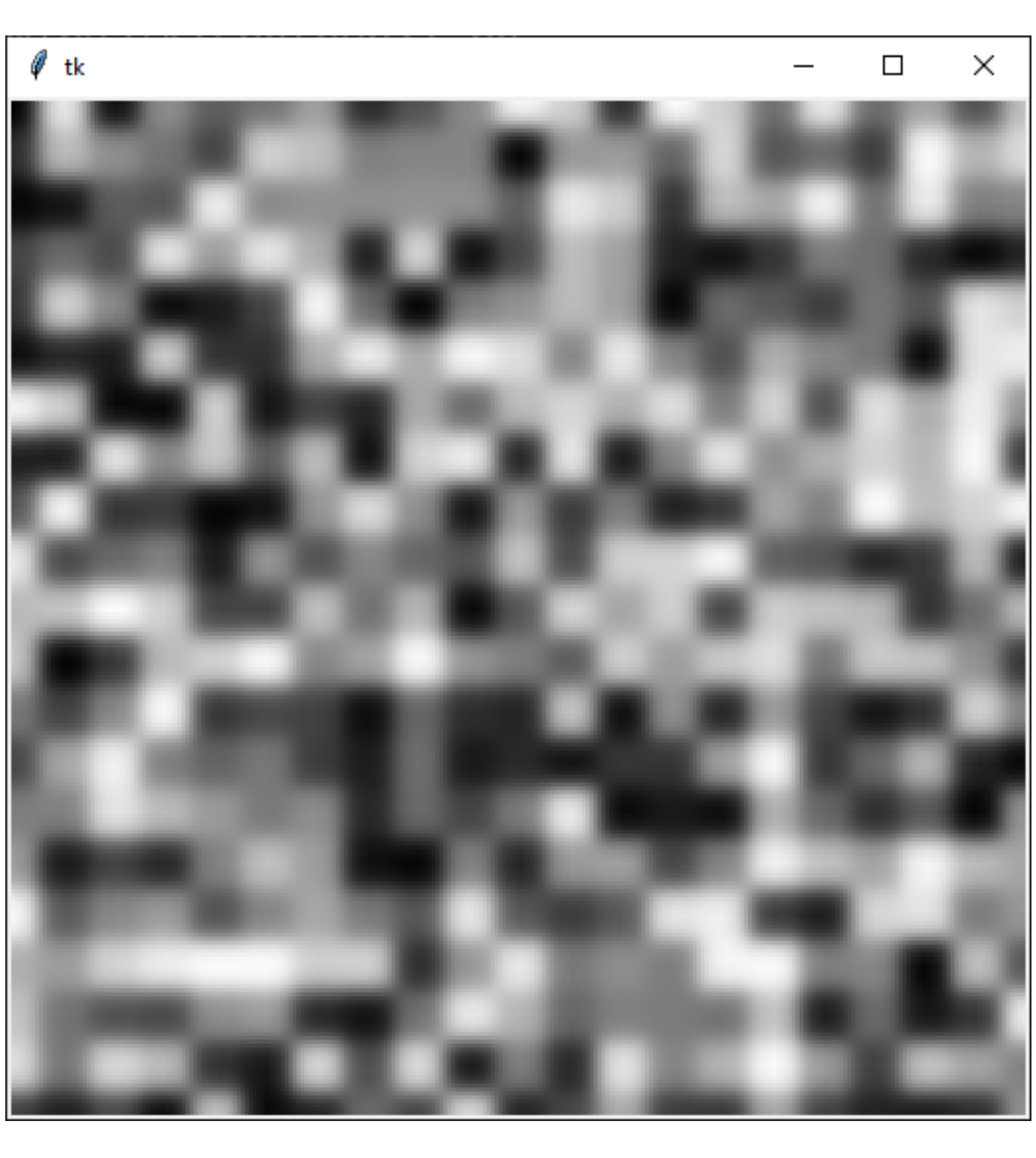
4.4. (0.2 балла)

Реализуйте функцию шума noise, которая по заданным координатам выдает случайное значение. Не используйте модуль random. Не забывайте, что глобальным состоянием пользоваться нельзя.



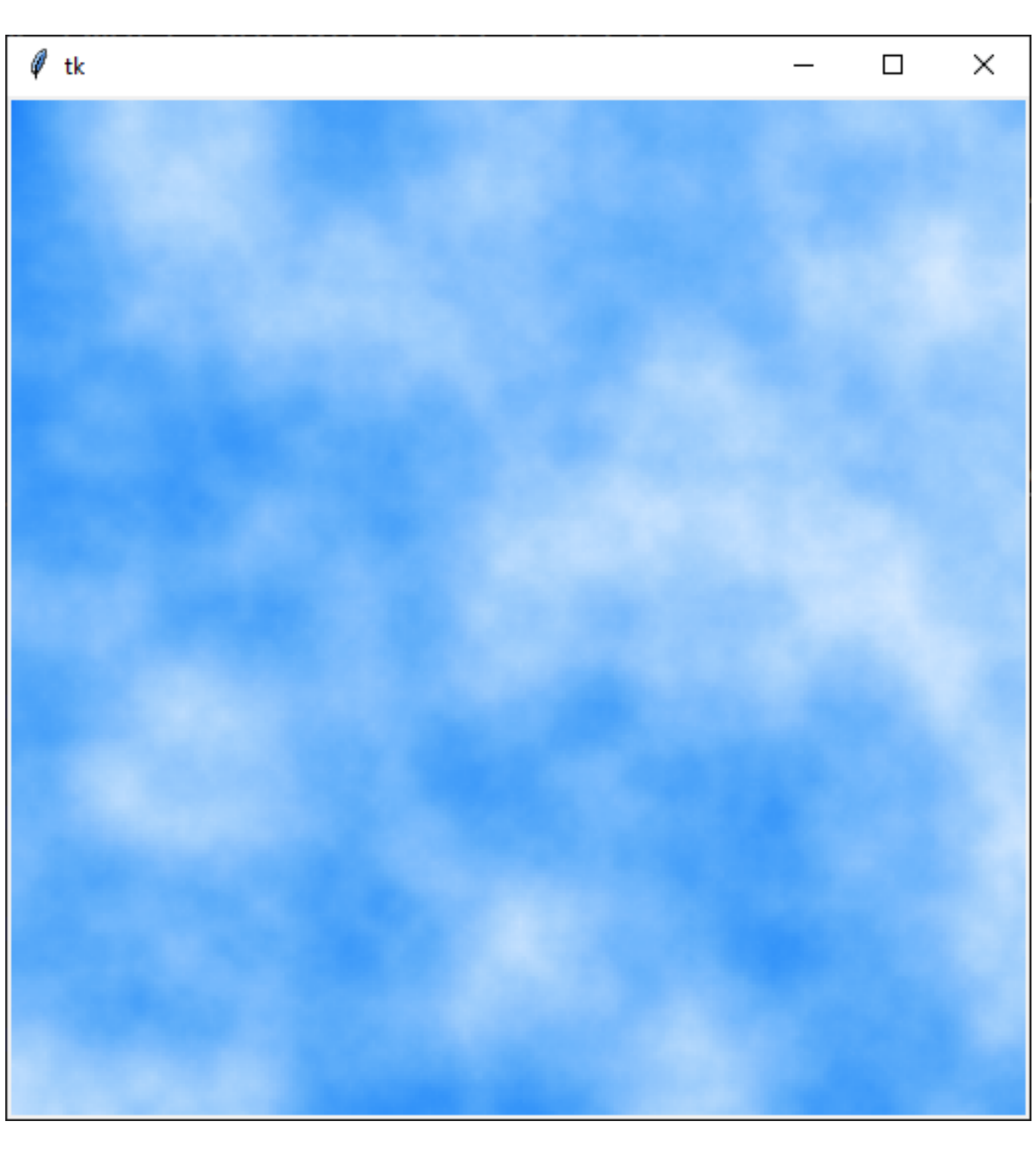
4.5. (0.4 балла)

Реализуйте функцию интерполяционного шума val_noise, используя алгоритм value noise (см. источники выше), а также ранее разработанную функцию noise.



4.6. (0.5 балла)

Изобразите облака с помощью алгоритма фрактального шума (fBm noise, см. источники выше) и с использованием ранее разработанной функции val_noise.



5. Алгоритмическая игра DandyBot

5.1. (0.4 балла за уровень)

Скачайте игру DandyBot: https://github.com/true-grue/DandyBot. Код для своего игрока записывается в файле user_bot.py. Игра запускается с помощью main.py.

Вот простой пример содержимого user_bot.py:

```
def script(check, x, y):
    return 'right'
```

Игровая логика записывается исключительно в теле функции script. В нашем случае игрок будет постоянно двигаться вправо.

Полный список действий, которые можно возвращать из функции script, задающей «интеллект» игрока:

- 'up'. Двигаться вверх на клетку.
- 'down'. Двигаться вниз на клетку.
- 'left'. Двигаться влево на клетку.
- 'right'. Двигаться вправо на клетку.
- 'pass'. Ничего не делать.
- 'take'. Взять золото.

Для изучения среды есть функция check:

- check('player', x, y). True, если какой-то игрок в позиции (x, y).
- check('gold', x , y). Если золото в позиции (x, y), то вернуть его количество, иначе вернуть 0.
- check('wall', x, y). True, если стена в позиции (x, y).
- check('level'). Вернуть номер текущего уровня.

Ваша задача — пройти все уровни. Дополнительно устанавливаемыми библиотеками и глобальными данными пользоваться нельзя.

Распределение сложных задач:

Пул задач 1: 3.7. + 3.8. + 3.9. - итого 0.9 балла.

Пул задач 2: с 4.1. по 4.6 - итого 1.9 баллов.

Пул задач 3: 5.1. - итого 2 балла.

Допускается сделать только 2 пула задач из предложенных на ваш выбор!