Зміст:

Задача:

Напишіть код, що генерує абсолютно випадкове число від 1 до 100… але без генератора випадкових чисел. Ви ж таке можете, чи не так?

Мова програмування: Python 3.11.5

Ідея:

Щоб отримати якесь рандомне число нам спочатку необхідно отримати псевдо рандомне число і після операцій над ним ми отримуємо число відповідно умови.

Для реалізації цього алгоритму зазвичай використовують time() но це скучно тому давайте використаємо рандомне число скажем як адресу об’єкта в памяті. Для цього створюєм об’єкт і отримуємо його адресу за допомогою id().

Наступна проблема це код в 1 рядок. Написати код в 1 рядок можна двома способами:

1 це використовуючи **;** для переносу но при цьому ми стикаємось з великими обмеженнями а це не зручно і скучно.

2 Це написати все як 1 дія. В python існують оператири or, and і ці оператири повертають об’єкт а не булеве значення (0 or 3 -> 3) на цій логіці і буде грунтуватись вся одна стрічка коду. З мінусів такий код майже не реально читати але в даному випадку це плюс.

Код:

```
print(not(_____:={0b1100100}) or _____.add(id) or _____.add(range) or _____.add(0b1) or not(_____:=list(_____)) or not (L:=0b1111101000) or not((___:= lambda: type('___', (), {'__': (lambda self: self)})())().__())  or ([id(____).__mod__(0b1100011) + 0b1 for ____ in [___().__() for ____ in _____[0](0b1111101000)]][L.__floordiv__(int:=_____[0b1].__add__(_____.__getitem__(1)))])+((_____[0b11](___().__()).__mod__(_____[int])).__truediv__(L))
```

Документація:

Частини:

```
print(not(_____:={0b1100100}) or _____.add(id) or _____.add(range) or _____.add(0b1) or not(_____:=list(_____)) or not (L:=0b1111101000) or not((___:= lambda: type('___', (), {'__': (lambda self: self)})())().__())  or ([id(_____).__mod__(0b1100011) + 0b1 for ____ in [___().__() for ____ in _____[0](0b1111101000)]][L.__floordiv__(int:=_____[0b1].__add__(_____.__getitem__(1)))])+((_____[0b11](___().__()).__mod__(_____[int])).__truediv__(L))
```

Частина 1:

Призначаємо змінні:

```
not(_____:={0b1100100}) or _____.add(id) or _____.add(range) or _____.add(0b1) or not(_____:=list(_____)) or not (L:=0b1111101000)
```

1) _____:={0b1100100}) - Створюємо змінну _____ типу set (множина, автоматично сортується і не має доступу за індексом), і додоємо в неї значення 0b1100100 (в десятковій системі = 100)
2) _____.add(id) - Добавляємо в set посилання на функцію id
3) _____.add(range) - Добавляємо в set посилання на функцію range
4) _____.add(0b1) - Добавляємо в set значення 0b1 (1)
5) (_____:=list(_____)) - Перетворюємо set в list для того щоб доступатись за індексом
6) (L:=0b1111101000) - Створюємо змінну L - куди записуємо значення 1000

Частина 2:

Створюємо клас:

```
or not((___:= lambda: type('___', (), {'__': (lambda self: self)})())().__())        or
```

Присутній лише 1 вираз і він самий важливий в коді оскільки він дозволяє створювати класи за допомогою лямбда функції

1) type - прекрасна функція яка дозволяє обійти обмеження класичного пайтона і створити об’єкт без конструкції класа, повертає екземпляр об’єкта.
2) '___', () - створюємо клас з назвою ___ який не наслідуються ні від яких інших об’єктів.
3) {'__': (lambda self: self)} - в середені класу створюємо метод __ який повертає екземпляр класа.

4) ())().__() - тут ми створюємо об’єкт і повертаємо його через функцію __ щоб or not повернуло false

Частина 3:

Отримуємо цілу частину:

([id(_____).__mod__(0b1100011) + 0b1 for _____ in [___().__() for _____ in _____[0](0b1111101000)]][L.__floordiv__(int:=_____[0b1].__add__(_____.__getitem__(1)))])

1) [___().__() for _____ in _____[0](0b1111101000)] - Тут ми створюємо масив з об’єктами щоб забити пам’ять, кількість елементів = 1000
2) [id(_____).__mod__(0b1100011) + 0b1 for _____ in [] - тут ми пробігаємось по цьому масиву і створюємо інший масив де вже записуємо id в пам’яті з котрого отримуємо остачу від ділення на 99 і після додаємо 1 (тобто число від 1 до 100 невключно)
3) [L.__floordiv__(int:=_____[0b1].__add__(_____.__getitem__(1)))]) - тут ми виконуємо цілочисельне ділення змінної L на змінну int яка вираховується додаванням двох елементів змінної _____ на першому індексі.
4) І останнє ми отримуємо масив і получаємо середину масиву, конструкція: array[index]

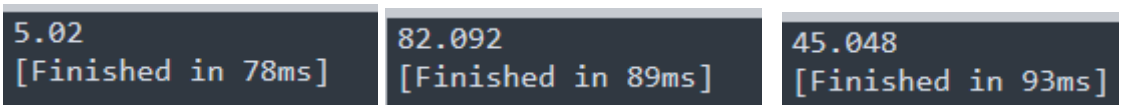
Частина 4:

Отримуємо дробову частину:

((_____[0b11](___().__()).__mod__(_____[int])).__truediv__(L))

1) (_____[0b11](___().__()) - отримуємо елемент з нашого списку зі змінними на 3 індексі (це функція id) і далі створюємо об’єкт повертаємо самого себе і отримуємо ід в пам’яті
2) __mod__(_____[int]) тут ми отримуємо остачу від ділення на елемент в масиві під індексом int (2 індекс це елемент 100)
3) __truediv__(L)) - це все націло на L (1000)

Приклад роботи:



Модифікуємо програму до функції

```
def a():

        return(not(_____:={0b1100100}) or _____.add(id) or _____.add(range) or  _____.add(0b1) or not(_____:=list(_____)) or not (L:=0b1111101000) or not((___:= lambda: type('___', (), {'__': (lambda self: self)})())().__()) or ([id(_____).__mod__(0b1100011) + 0b1 for _____ in [___().__() for _____ in _____[0](0b1111101000)]] [L.__floordiv__(int:=_____[0b1].__add__(_____.__getitem__(1)))]) + ((_____[0b11](___().__()).__mod__(_____[int])).__truediv__(L)))


l = list()

for i in range(500):

        l.append(a())


l.sort()

print(l)
```

Вивід масиву при 500 ітераціях наступний:

```
[1.052, 1.06, 1.088, 2.02, 2.032, 2.044, 2.048, 2.08, 3.0, 3.004, 3.008, 3.016, 3.02, 3.044, 3.048, 3.08, 3.08, 3.092, 4.028, 4.032, 4.06, 4.06,
4.068, 4.084, 5.02, 5.028, 5.036, 5.048, 5.056, 6.008, 6.044, 6.048, 6.06, 6.08, 7.032, 7.052, 7.076, 8.008, 8.016, 8.02, 8.044, 8.048, 8.072,
8.076, 8.088, 9.028, 9.032, 9.032, 9.076, 10.004, 10.028, 10.064, 10.072, 10.072, 10.092, 10.092, 11.0, 11.008, 11.096, 12.076, 12.076, 12.096,
13.008, 13.012, 14.02, 14.02, 14.036, 14.04, 14.04, 14.092, 15.008, 15.028, 15.048, 15.076, 15.092, 16.008, 16.008, 16.028, 16.028, 16.032, 16.044,
16.048, 16.076, 16.084, 16.084, 17.012, 17.012, 17.02, 17.056, 17.064, 17.08, 18.024, 18.028, 18.052, 19.008, 19.048, 19.072, 19.084, 20.016,
20.024, 20.044, 20.048, 20.052, 20.064, 20.064, 20.068, 20.068, 20.076, 20.084, 20.088, 21.028, 21.032, 22.008, 22.008, 22.028, 22.044, 22.044,
22.072, 22.084, 23.016, 23.024, 23.028, 23.092, 24.0, 24.012, 24.02, 24.044, 24.048, 24.064, 24.076, 24.088, 24.088, 25.0, 25.068, 26.044, 26.048,
26.052, 26.068, 26.092, 27.004, 27.076, 27.092, 28.004, 28.016, 28.032, 29.016, 29.028, 29.084, 30.0, 30.036, 30.06, 30.064, 30.08, 31.004, 31.012,
31.024, 31.024, 31.032, 32.008, 32.024, 32.028, 32.056, 33.004, 33.044, 33.084, 34.004, 34.044, 34.068, 34.072, 34.076, 34.088, 35.008, 35.012,
35.072, 35.076, 36.004, 36.024, 36.028, 36.04, 36.06, 36.06, 36.072, 36.096, 37.016, 37.028, 37.052, 37.052, 38.02, 38.084, 38.096, 39.0, 39.02,
39.036, 39.052, 39.052, 39.06, 39.06, 39.06, 39.06, 39.064, 39.072, 39.096, 40.004, 40.012, 40.024, 40.04, 40.044, 40.076, 40.092, 41.004, 41.024,
41.052, 41.088, 42.0, 42.064, 42.08, 43.008, 43.024, 43.028, 43.036, 43.044, 43.052, 43.06, 43.072, 43.076, 43.076, 43.092, 44.008, 44.028, 44.088,
45.004, 45.016, 45.052, 45.056, 45.08, 46.004, 46.044, 46.052, 46.076, 46.096, 47.004, 47.004, 47.028, 47.028, 47.064, 47.076, 47.08, 47.096,
47.096, 48.008, 48.024, 49.008, 49.016, 49.06, 49.064, 49.096, 49.096, 50.012, 50.02, 50.032, 50.04, 50.052, 51.0, 51.0, 51.004, 51.016, 51.028,
51.052, 51.08, 51.084, 51.088, 51.092, 52.008, 52.016, 52.028, 52.032, 52.068, 53.0, 53.008, 53.028, 53.068, 54.004, 54.016, 54.056, 54.068, 55.0,
55.028, 55.084, 56.024, 56.048, 56.056, 56.056, 56.072, 56.084, 57.0, 57.012, 57.012, 57.048, 57.052, 57.076, 57.092, 58.004, 58.02, 58.048,
58.092, 59.052, 59.064, 59.096, 60.0, 60.048, 60.052, 60.08, 61.0, 61.012, 61.032, 61.04, 61.044, 61.044, 61.052, 61.088, 61.096, 62.048, 62.052,
62.084, 63.008, 63.012, 63.04, 63.044, 63.048, 63.084, 64.024, 64.032, 64.04, 65.02, 65.056, 65.06, 65.064, 65.084, 66.004, 66.012, 66.02, 66.032,
66.04, 66.044, 66.048, 66.056, 66.076, 67.04, 67.044, 67.072, 68.04, 68.06, 68.084, 69.004, 69.052, 69.056, 69.072, 69.076, 69.088, 69.096, 70.072,
70.08, 71.012, 71.032, 71.048, 71.088, 72.012, 72.016, 72.016, 72.02, 72.024, 72.04, 72.04, 72.096, 73.0, 73.032, 73.084, 74.0, 74.064, 74.072,
74.076, 75.008, 75.024, 75.048, 75.06, 75.072, 75.088, 75.092, 75.096, 76.0, 76.016, 76.052, 76.068, 76.084, 76.096, 77.004, 77.016, 77.036,
77.036, 77.036, 77.064, 78.016, 78.092, 79.008, 79.056, 79.06, 80.016, 80.04, 80.088, 81.004, 81.044, 81.048, 81.06, 81.08, 81.084, 82.032, 82.032,
82.056, 82.064, 82.08, 82.084, 83.028, 83.048, 83.06, 84.02, 84.032, 84.048, 84.048, 84.056, 85.012, 85.028, 85.088, 85.092, 86.0, 86.004, 86.016,
86.024, 86.028, 86.06, 86.088, 87.004, 87.016, 87.04, 87.056, 87.076, 88.004, 88.016, 88.048, 88.092, 89.004, 89.032, 89.06, 90.048, 90.06, 90.068,
90.072, 90.076, 90.092, 91.06, 91.068, 91.096, 92.024, 92.048, 92.048, 92.068, 92.068, 93.004, 93.044, 93.056, 93.056, 93.088, 93.096, 94.008,
94.02, 94.024, 94.048, 94.076, 95.0, 95.072, 95.084, 95.096, 96.028, 96.032, 96.04, 96.052, 96.076, 97.008, 97.02, 97.068, 97.084, 97.092, 98.036,
98.048, 98.076, 98.084, 99.012, 99.04, 99.052, 99.06, 99.076]
```

Компіляція:

В репозиторій я додав екзешний файл скомпільованої програми, для його запуску просто відкрите консоль і перетяніть шлях до файлу туди.

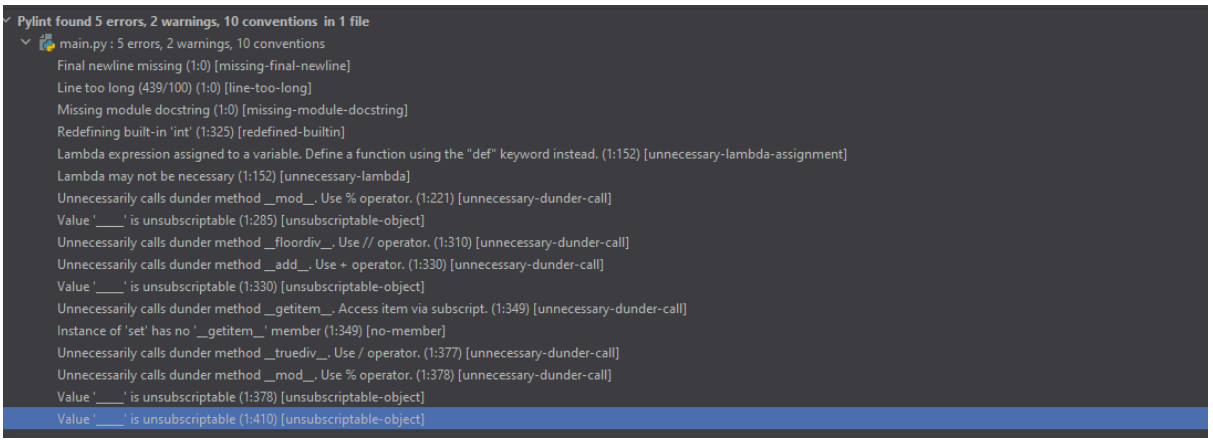
Якщо ви хочете самі скомпілювати то вам необхідно використати pyinstaller.

Якщо ви хочете запустити на інтерпретаторі пайтона то необхідно використовувати версію яка підтримує оператор **:=**

Заключення (непотріб):

Скажу так, це не найгірше що можна написати, можна було б додати дофіга непотрібних дій які в кінечному результаті вертали необхідне нам значення, але мені на це шкода часу і це важко документувати.

Ну а всіми любимий pylint (Програма для перевірки якості коду за стандартом pep8) видає таку штукню:



І це при тому що вона не може аналізувати змінні і їх назви оскільки ми все виконуємо в функції print.

Для себе відзначу що я гордий шо написав таку гидоту, це ужас, писати до цього документацю, а ще сидіти і намагатися зрозуміти шо я тут написав це просто ужас. Навіть коли я тільки починав вчитись кодити я не писав на стільки ужасно. В загальному задачка цікава і потрібно було подумати як зробити все хоть чучуть оригінально.

В загальному я сам для себе хотів зробити подібний челендж і тут підвернулась нагода. Швидше всього ця робота як і ця документація кануть в лету як всі інші початі і завершені проекти, що ж, це було весело і як мінімум я задовільнив своє бажання написати ужасний код. Якщо ви все ще це читаєте то бажаю вам хорошого дня, і на цьому пора рухатись далі.