Урок 9. Повторение. Разнобой.

ФИО_____

Тест:

1. *Что выведет данная программа?*

```
a = [1, 2, 3]
b = a[:]
b[0] = 99
print(a, b is a)
```

2. *Что верно про tuple?*
   A) Можно добавлять элементы методом append
   B) Может быть ключом словаря
   C) Изменяемый как список
   D) Нельзя распаковывать со *
3. *Что произойдет?*

```
import json
print(json.dumps({"x": {1,2,3}}))
```

_____

4. *Что верно про аннотации?*
   A) Они принудительно проверяются интерпретатором
   B) Это подсказки; доступны в func.__annotations__
   C) Нельзя аннотировать возвращаемое значение
   D) Разрешены только встроенные типы, нельзя из typing

5. *Что выведет данная программа?*

```
x = 10
def outer():  1 usage
    x = 5
    def inner():
        nonlocal x
        x += 1
        return x
    return inner()
```

_____

6. *Каким будет порядок применения декораторов?*

@A
@B

def f(): ...

7. *Что актуально про @classmethod?*
   A) Первый аргумент — self
   B) Первый аргумент — класс (cls)
   C) Нельзя вызывать через класс
   D) Работает только на экземпляре
8. *Что выведет данная программа?*

```python
class C:  2 usages
    name = "base"
    @classmethod  2 usages
    def get(cls):
        return cls.name

class D(C):  1 usage
    name = "child"

print(C.get(), D.get())
```

9. *Что выведет данная программа?*

```python
class Flag:  1 usage
    def __len__(self):
        return 0
print(bool(Flag()))
```

10. *Как сделать свойство только для чтения x?*
    A) __getattr__
    B) @property без сеттера
    C) @staticmethod
    D) global x
11. *Что выведет данная программа? Приведите пример вывода через print().*

```python
class V:  3 usages
    def __init__(self,x):
        self.x = x
    def __add__(self, other):
        return V(self.x + other.x)
    def __repr__(self):
        return f"V({self.x})"
```

12. *Как корректно расширить поведение метода базового класса?*
A) Переопределить метод и не вызывать базовый
B) Вызвать super().method(...) внутри переопределённого метода
C) Изменить метод базового класса на лету
D) Вызвать метод напрямую через имя базового класса без self

Теория:

Шпаргалка:

- **Коллекции**:
  list — изменяемые, порядок; tuple — неизменяемые; set — уникальность и быстрый in; срезы создают копии; копирование вложенных структур требует copy.deepcopy.
- **Словари/JSON**:
  d = {} / dict(), безопасное чтение d.get(k, default), группировка через setdefault. JSON: ключи — строки; json.dumps/loads.
- **Функции**:
  сигнатуры, передача ссылок на объекты; избегаем изменяемых значений по умолчанию, global, nonlocal; аннотации — подсказки.
- **Декораторы**:
  «обёртка» вокруг функции; *args/**kwargs; параметризованные — фабрика → декоратор → wrapper.
- **ООП (базовое)**:
  классы, __init__, атрибуты класса/экземпляра, @classmethod/@staticmethod.
- **Принципы ООП**:
  инкапсуляция (соглашения _x, @property), наследование/super(), полиморфизм (единый интерфейс для разных типов), абстракция (abc.ABC).
- **Магические методы**:
  перезагрузка операторов, методы (__add__/__len__/__bool__)

Задачи:

1. Дан кортеж из шести цифр. Проверьте, можно ли собрать из этих цифр счастливый билет: если сумма цифр, стоящих на чётных местах билета, равна сумме цифр, стоящих на нечётных местах. Например, 350053 (3 + 0 + 5 = 5 + 0 + 3).
2. Напишите функцию, которая возвращает словарь *буква –> список слов*, где буква — это первый символ слова в нижнем регистре. В списках сохраняйте порядок появления. Пустые строки игнорируйте.
   Сигнатура: def index_by_first_letter(words: list[str]) -> dict[str, list[str]]:
   Пример: ["Apple", "art", "Bee"] -> {"a": ["Apple","art"], "b": ["Bee"]}.
3. Сформируйте список кортежей из соседних элементов: (nums[0],nums[1]), (nums[2],nums[3]), … Если длина нечётная — последний элемент **игнорируйте**.
   Сигнатура: def pairwise(nums: list[int]) -> list[tuple[int, int]]:
   Пример: [10,20,30,40,50] -> [(10,20),(30,40)].
4. Напишите функцию, которая принимает любые позиционные и именованные значения. Верните словарь тип -> список значений этого типа:
   - учитывать **все** значения из *args и **все** значения из **kwargs (ключи не группировать);

- порядок внутри списков — по порядку поступления;
- пустой ввод → {}.

Сигнатура: def bucket_by_type(*args, **kwargs) -> dict[type, list]:
Пример: bucket_by_type(1, "a", 2.0, flag=True, name="x")
{int:[1], str:["a","x"], float:[2.0], bool:[True]}
Подсказка: используйте функцию type() и обращение к именованным элементам через kwargs.values().

5. Напишите декоратор @time_budget(ms): если функция выполнялась дольше ms, напечатать предупреждение "slow" и вернуть результат как есть (без убийства процесса). Использовать time.perf_counter из библиотеки time.
6. Строки в Python сравниваются на основании значений символов из таблицы кодировки. Т.е. если мы захотим выяснить, что больше: «Apple» или «Яблоко», – то «Яблоко» окажется бОльшим по значениям из таблицы кодировки для каждой буквы.
   Создайте класс RealLenString и перезагрузите в нем магические методы сравнения таким образом, чтобы они сравнивали посимвольно, а не по кодировке каждой буквы. Решите задачу двумя способами, второй способ сократит количество методов класса до трех (включая __init__), с помощью декоратора @total_ordering из библиотеки functools**.**
7. Создайте абстрактный класс Role с методом can(action: str) -> bool.
   Классы:

- Admin (всё может), Editor (create/update/read), Viewer (read только).
  Списки разрешённых действий держите **инкапсулированно** (например, приватное поле).
- Класс User(name: str, role: Role).
  Функция allowed(users: list[User], action: str) -> list[str] — вернуть имена пользователей, которым разрешено действие; не использовать проверки типа.
  Пример: allowed([User("Ann", Viewer()), User("Bob", Admin())], "delete") -> ["Bob"]