1. from itertools import product

این خط برای وارد کردن ماژول product از کتابخانه itertools استفاده می‌شود. این ماژول به ساخت تمام ترکیب‌های ممکن از مجموعه‌های ورودی کمک می‌کند. در اینجا برای ترکیب‌های ممکن کلاس‌ها، زمان‌ها و اساتید استفاده می‌شود.

---

2. from collections import defaultdict

این خط برای وارد کردن ماژول defaultdict از کتابخانه collections است defaultdict .یک ساختار داده‌ای است که در صورت نیاز کلیدهای جدید را ایجاد می‌کند و مقدار پیش‌فرض را تنظیم می‌کند. در اینجا از آن برای ایجاد دامنه‌های جدید استفاده می‌شود.

3.تعریف instructors

```
instructors = {
    "AI": ["Dr.Moosavi", "Dr.Shahabi"],
    "Physics": ["Dr.Pouzesh"],
    "Chemistry": ["Dr.Fathi"],
    "Music": ["Dr.Shokoohi", "Dr.Mortazavi"],
    "Cinema": ["Dr.Mortazavi", "Dr.Khosravani"],
    "Algebra": ["Dr.Pourbagheri"]
}
```

این بخش تعریف اساتید مختلف برای هر درس است. هر درس یک لیست از اساتید مرتبط با خود دارد.

4.تعریف classrooms

```
classrooms = ["Room1", "Room2", "Room3"]
```

این لیست شامل مکان‌های مختلف برای برگزاری کلاس‌هاست.

5.تعریف time_slots

```
time_slots = ["9:00-10:00", "10:00-11:00", "11:00-12:00"]
```

این لیست بازه‌های زمانی در دسترس برای برگزاری کلاس‌ها را مشخص می‌کند.

---

6. generate_domains

```
def generate_domains():
    domains = {}
    for course, teachers in instructors.items():
        domains[course] = list(product(classrooms, time_slots, teachers))
    return domains
```

این تابع دامنه‌هایی برای هر درس ایجاد می‌کند که شامل تمام ترکیب‌های ممکن از کلاس‌ها، زمان‌ها و اساتید است. با استفاده از product، تمام ترکیب‌های ممکن برای یک درس ساخته می‌شود.

---

7. is_consistent

```
def is_consistent(assignment, course, value):
    room, time, teacher = value
    for assigned_course, (assigned_room, assigned_time, assigned_teacher) in assignment.items():
        if room == assigned_room and time == assigned_time:
            return False
        if teacher == assigned_teacher and time == assigned_time:
            return False
    return True
```

این تابع بررسی می‌کند که آیا یک مقدار برای یک درس خاص، با سایر تخصیص‌ها سازگار است یا خیر. اگر در یک زمان و مکان یا با همان استاد دیگر درس‌هایی اختصاص داده شده باشد، ناسازگار است و Falseبازمی‌گرداند.

---

8. select_unassigned_variable

```
def select_unassigned_variable(domains, assignment):
    unassigned = [var for var in domains if var not in assignment]
    return min(unassigned, key=lambda var: len(domains[var]))
```

این تابع، متغیری که هنوز تخصیص داده نشده را انتخاب می‌کند. برای انتخاب مناسب‌ترین متغیر، از متغیری استفاده می‌شود که دارای دامنه کوچکتر باشد.

---

9. order_domain_values

```
def order_domain_values(variable, domains, assignment):
    def count_conflicts(value):
        room, time, teacher = value
        conflicts = 0
        for var in domains:
            if var in assignment:
                continue
            for val in domains[var]:
                if val[0] == room and val[1] == time:
                    conflicts += 1
                if val[2] == teacher and val[1] == time:
                    conflicts += 1
        return conflicts

    return sorted(domains[variable], key=count_conflicts)
```

این تابع با استفاده از **LCV (Least Constraining Value)**، دامنه مقادیر را براساس تعداد محدودیت‌هایی که ایجاد می‌کنند مرتب می‌کند. به این معنا که مقادیری که کمترین تعداد ناسازگاری (conflict) دارند، در اولویت قرار می‌گیرند.

---

10. forward_checking

```
def forward_checking(domains, variable, value):
    room, time, teacher = value
    new_domains = defaultdict(list, {var: list(vals) for var, vals in domains.items()})

    for var in domains:
        if var == variable:
            continue
        new_domains[var] = [
            val for val in domains[var]
            if (val[0] != room or val[1] != time) and
               (val[2] != teacher or val[1] != time)
        ]

        if not new_domains[var]:
            return None

    return new_domains
```

این تابع عملیات **Forward Checking**را انجام می‌دهد. با بررسی هر دامنه متغیر دیگر، مقادیر ناسازگار با تخصیص فعلی حذف می‌شوند. در صورتی که یک دامنه کاملاً خالی شود، نشان‌دهنده ناسازگاری است و None باز می‌گردد.

---

11. backtrack

```
def backtrack(assignment, domains):
    if len(assignment) == len(instructors):
        return assignment

    variable = select_unassigned_variable(domains, assignment)
    for value in order_domain_values(variable, domains, assignment):
        if is_consistent(assignment, variable, value):
            assignment[variable] = value
            new_domains = forward_checking(domains, variable, value)

            if new_domains is not None:
                result = backtrack(assignment, new_domains)
                if result:
                    return result
```

```
        del assignment[variable]

    return None
```

این تابع عملیات اصلی برنامه‌ریزی (Backtracking) را انجام می‌دهد. ابتدا متغیری که هنوز تخصیص داده نشده است انتخاب شده و مقادیر آن با استفاده از LCV مرتب می‌شوند. سپس هر مقدار به بررسی سازگاری می‌پردازد و در صورت سازگاری، تخصیص داده می‌شود. در صورت ناسازگاری، برگشتی انجام می‌شود و متغیر پاک می‌شود تا یک ترکیب دیگر بررسی شود.

---

12. solve_schedule

```
def solve_schedule():
    domains = generate_domains()
    assignment = {}
    result = backtrack(assignment, domains)
    return result
```

این تابع برای حل کلی مسأله زمان‌بندی استفاده می‌شود. ابتدا دامنه‌ها تولید می‌شوند و سپس عملیات Backtracking برای یافتن زمان‌بندی مناسب اجرا می‌شود.

13.اجرای solve_schedule

```
solution = solve_schedule()

if solution:
    print("Scheduling Solution:")
    for course, (room, time, teacher) in solution.items():
        print(f"{course}: {room} at {time} with {teacher}")
else:
    print("No solution found!")
```

در اینجا نتیجه به صورت بصری چاپ می‌شود. اگر نتیجه‌ای یافت شود، زمان‌بندی ارائه می‌شود و در غیر این صورت اعلام می‌شود که هیچ راه‌حلی پیدا نشده است.