مذكرة مراجعة عملية

Python OOP

البرمجة كائنية التوجه في Python

تراجع Classes وObjects وEncapsulation وInheritance وباقي أساسيات OOP من خلال كود قصير ومشروع واقعي، ثم تختبر نفسك بإجابات موثقة بالناتج.

Python 3.12 | OOP + Project | 35 Pages

إعداد
محمود منصور
Mahmoud Mansour

المنصة التعليمية
khatwa tech
youtube.com/@khatwatech

{DATE}

دليل الاستخدام

خريطة المذكرة

اقرأ الفكرة؛ جرّب الكود؛ وبعدها اختبر نفسك.

الموضوع	الصفحات
01. OOP Introduction	3–4
02. Classes & Objects	5–7
03. Constructors & Attributes	8–9
04. Encapsulation & Properties	10–12
05. Inheritance	13–15
06. Polymorphism	16–17
07. Abstraction	18–19
08. Special Methods	20–21
09. OOP Design & Dataclass	22–23
10. Course Management Project	24–26
الأخطاء + ورقة مراجعة	27–28
6 توقع الناتج + 6 تمارين	29–30
الإجابات والحالات الحدّية	31–34
التوثيق والخطوة التالية	35

طريقة استخدام المذكرة في 3 نقاط

- اقرأ تعريف الفكرة ثم جرب كتابة الكود من غير نسخ حرفي.
- قبل مشاهدة الناتج، توقعه ثم شغل البرنامج وقارن.
- حل صفحة التمارين من غير مساعدة وبعدها راجع الحلول والحالات الحدّية.

الإصدار المحتوى مستهدف Python 3.12. تم فحص بناء الكود وفق قواعد 3.12، وتشغيل الأمثلة على Python 3.13.5 باستخدام ميزات متوافقة، وليس على Interpreter 3.12 نفسه.

مقدمة OOP

01. Why OOP?

بننظّم البيانات والسلوكيات المتصلة في Object واحد.

في الـProcedural style بتفصل البيانات عن الدوال. في OOP، الـClass بتحدد شكل البيانات والـMethods اللي بتتعامل معاها. مثال: طالب مع اسمه ودرجته ودالة تحدد النجاح.

Procedural	Object-Oriented
Dictionary ودالة منفصلة	Object يحتوي البيانات والـMethods
مناسب للمسائل الصغيرة	مفيد لما توجد Entities كثيرة
أقل تنظيمًا عند التوسع	أسهل في إعادة الاستخدام والتطوير

نفس الفكرة باستخدام Class

PYTHON

```python
class Student:
    def passed(self, grade):
        return grade >= 50

student = Student()
print(student.passed(82))
```

الناتج الفعلي

```
True
```

الفرق اللي يهمك الـClass قالب؛ الـObject نسخة منه. استخدام OOP مش إلزامي لكل برنامج.

مقدمة OOP

OOP: المبادئ الأربعة

افهم أسماء المبادئ قبل التطبيق العملي.

المبدأ	الفكرة العملية
Encapsulation	تتحكم في قراءة وتعديل البيانات
Inheritance	Class جديدة تعيد استخدام سلوك Class أساسية
Polymorphism	واجهة مشتركة مع سلوك مختلف حسب الـObject
Abstraction	توضح العمليات المطلوبة وتخفي تفاصيل التنفيذ

Object يتصرف حسب بياناته

PYTHON

```python
class Course:
    def __init__(self, title, price):
        self.title = title
        self.price = price

    def label(self):
        return f"{self.title}: {self.price}"

print(Course("Python", 300).label())
```

الناتج الفعلي

```
Python: 300
```

انتبه الـOOP مش مجرد كتابة class. أهميته في تصميم مسؤوليات واضحة والبعد عن تكرار نفس المنطق.

Classes & Objects

Classes & Objects .02

الـClass تعريف، والـObject نسخة مستقلة.

اكتب `class` متبوعة باسم بأسلوب PascalCase؛ أنشئ Object باستدعاء اسم الـClass. الـAttributes بتخزن البيانات، والـMethods بتنفذ سلوك.

إنشاء أول Object

PYTHON

```python
class Course:
    category = "Programming"

course = Course()
print(course.category)
print(type(course).__name__)
```

الناتج الفعلي

```
Programming
Course
```

تعريف بيانات على Object

PYTHON

```python
class Student:
    pass

student = Student()
student.name = "Ali"
student.grade = 90
print(student.name, student.grade)
```

الناتج الفعلي

```
Ali 90
```

خطأ مشهور تقدر تضيف Attribute بعد الإنشاء، لكن الأفضل تعريف البيانات الأساسية في `__init__()` ؛ ده بيخلي شكل الـObject متوقع.

Classes & Objects

Objects: كل نسخة مستقلة

التعديل على بيانات طالب لا يغيّر بيانات الطالب الثاني.

Instance Attributes مختلفة

PYTHON

```python
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

first = Student("Ali", 80)
second = Student("Sara", 95)
first.grade = 88
print(first.grade)
print(second.grade)
```

الناتج الفعلي

```
88
95
```

المصطلح	معناه
Instance	Object حقيقية متكوّنة من Class
Instance Attribute	قيمة تخص Object معينة
Class Attribute	قيمة محفوظة على الـClass وقد تشاركها النسخ

جرّب بنفسك كوّن Object ثالثة باسم Mona ودرجة 70، وعدّل درجتها من غير التأثير على باقي الطلاب.

Classes & Objects

Methods & self

`self` بيشير للـObject اللي استدعت الـMethod.

Method تقرأ بيانات Object

PYTHON

```python
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

    def result(self):
        return "Passed" if self.grade >= 50 else "Failed"

print(Student("Mona", 75).result())
print(Student("Omar", 40).result())
```

الناتج الفعلي

```
Passed
Failed
```

عند استدعاء `student.result()`، Python بتمرر `student` تلقائيًا كأول Argument للـMethod. اسم `self` مجرد Convention متّبع لازم نلتزم به للوضوح.

انتبه لو نسيت `self` في تعريف Instance Method، غالبًا هتشوف `TypeError` وقت الاستدعاء.

Constructors & Attributes

03. Constructors: `__init__`

تجهّز بيانات الـObject لحظة إنشائها.

تهيئة بيانات الطالب

PYTHON

```python
class Student:
    def __init__(self, name, grade=0):
        self.name = name
        self.grade = grade

first = Student("Ali")
second = Student("Sara", 93)
print(first.name, first.grade)
print(second.name, second.grade)
```

الناتج الفعلي

```
Ali 0
Sara 93
```

`__init__()` هي Initializer مش Method لإنشاء الـObject نفسه، والـDefault Arguments مفيدة لما القيمة اختيارية. خليها بسيطة: تهيئة وتحقق أساسي فقط.

خطأ مشهور بلاش تستخدم List قابلة للتعديل كـDefault Argument. استخدم `None` ثم أنشئ List جديدة داخل الدالة.

Constructors & Attributes

Instance vs Class Attributes

افهم إمتى الخاصية بتكون مشتركة وإمتى تخص نسخة.

Class Attribute وقيمة خاصة لكل نسخة

PYTHON

```python
class Course:
    platform = "khatwa tech"

    def __init__(self, title):
        self.title = title

first = Course("Python")
second = Course("Java")
print(first.platform, first.title)
print(second.platform, second.title)
```

الناتج الفعلي

```
khatwa tech Python
khatwa tech Java
```

تعديلات على Class Attribute

PYTHON

```python
class Course:
    category = "Programming"

one = Course()
two = Course()
Course.category = "Technology"
print(one.category)
print(two.category)
```

الناتج الفعلي

```
Technology
Technology
```

الفرق اللي يهمك استخدم Class Attribute للثوابت المشتركة؛ وInstance Attribute لحالة تخص Object واحدة.

Encapsulation

Encapsulation .04

حافظ على صحة البيانات بدل تعديلها من أي مكان بلا قواعد.

في Python لا يوجد `private` صارم مثل بعض اللغات. الاسم `_balance` إشارة إنه Internal. الاسم `__name` يستخدم Name Mangling لتقليل التعارض، مش حماية أمنية.

تعديل آمن خلال Method

PYTHON

```python
class Wallet:
    def __init__(self):
        self._balance = 0

    def add(self, amount):
        if amount > 0:
            self._balance += amount

    def balance(self):
        return self._balance

wallet = Wallet()
wallet.add(80)
wallet.add(-5)
print(wallet.balance())
```

الناتج الفعلي

```
80
```

انتبه الشرطة السفلية `_` Convention وليست منعًا تقنيًا للوصول. لا تستخدمها بدل التحقق من المدخلات.

Encapsulation

`@property` والتحقق

اكتب Validation في مكان واحد، واستخدم Syntax يشبه Attribute عادية.

Property بحدود للدرجات

PYTHON

```python
class Student:
    def __init__(self, grade):
        self.grade = grade

    @property
    def grade(self):
        return self._grade

    @grade.setter
    def grade(self, value):
        if not 0 <= value <= 100:
            raise ValueError("Invalid grade")
        self._grade = value

student = Student(75)
student.grade = 90
print(student.grade)
```

الناتج الفعلي

```
90
```

خطأ مشهور في Setter، خزّن القيمة في `_grade` وليس `self.grade` ؛ وإلا هتستدعي الـSetter بلا نهاية.

Encapsulation

Properties & Method Types

اختر النوع المناسب حسب احتياج Method للـObject أو الـClass.

النوع	أول Argument	متى؟
Instance Method	`self`	تقرأ أو تعدّل بيانات Object
Class Method	`cls`	تشتغل مع إعدادات Class أو Factory
Static Method	لا يوجد تلقائي	Utility مرتبطة بالـClass منطقيًا
`@property`	`self`	حساب أو وصول مضبوط لقيمة

Class Method وStatic Method

PYTHON

```python
class Price:
    tax_rate = 0.10

    @classmethod
    def tax(cls, amount):
        return amount * cls.tax_rate

    @staticmethod
    def valid(amount):
        return amount >= 0

print(Price.tax(200))
print(Price.valid(-2))
```

الناتج الفعلي

```
20.0
False
```

الفرق اللي يهمك اختيار Method العادية هو الافتراضي. استخدم Decorator لما تكون ليها فائدة واضحة.

Inheritance

Inheritance .05

الـChild Class بترث سلوك الـParent وتقدر تضيف عليه.

Student يرث من Person

PYTHON

```python
class Person:
    def __init__(self, name):
        self.name = name

    def greet(self):
        return f"Hi {self.name}"

class Student(Person):
    pass

print(Student("Ali").greet())
```

الناتج الفعلي

```
Hi Ali
```

مصطلح	معناه
Parent / Base	Class أساسية
Child / Derived	Class وارثة
Inherited Method	Method متاحة للابن بدون إعادة كتابتها
Override	كتابة نسخة مختلفة من Method موروثة

انتبه Inheritance مناسبة لعلاقة `is-a` مثل Student is a Person؛ مش لمجرد تجنب نسخ الكود.

Inheritance

`super()` وإضافة سلوك

تقدر تستفيد من تهيئة الأب وبعدين تضيف Attributes جديدة.

استدعاء Constructor للأب

PYTHON

```python
class Person:
    def __init__(self, name):
        self.name = name

class Instructor(Person):
    def __init__(self, name, subject):
        super().__init__(name)
        self.subject = subject

teacher = Instructor("Mona", "Python")
print(teacher.name, teacher.subject)
```

الناتج الفعلي

```
Mona Python
```

Overriding مع super

PYTHON

```python
class Person:
    def label(self):
        return "Person"

class Student(Person):
    def label(self):
        return super().label() + " -> Student"

print(Student().label())
```

الناتج الفعلي

```
Person -> Student
```

جرّب بنفسك اضف خاصية years_experience للـInstructor واطبعها مع name.

Inheritance

Multiple Inheritance & MRO

ترتيب البحث عن Method بيهم لما يوجد أكثر من Parent.

ترتيب البحث عن Method

PYTHON

```python
class A:
    def describe(self):
        return "A"

class B:
    def describe(self):
        return "B"

class C(A, B):
    pass

print(C().describe())
print([cls.__name__ for cls in C.__mro__])
```

الناتج الفعلي

```
A
['C', 'A', 'B', 'object']
```

MRO اختصار Method Resolution Order؛ تقدر تتعرف على الترتيب باستخدام `Class.__mro__`. في أغلب المشاريع المبتدئة خلي الوراثة بسيطة وفضّل Composition لما العلاقة `has-a`.

خطأ مشهور Multiple Inheritance ممكن تصعّب فهم الكود؛ متستخدمهاش إلا لو عندك سبب تصميمي واضح.

Polymorphism

Polymorphism .06

نفس اسم Method يقدّم سلوك مختلف حسب الـObject.

Override لعرض دور مختلف

PYTHON

```python
class User:
    def role(self):
        return "User"

class Admin(User):
    def role(self):
        return "Admin"

for account in [User(), Admin()]:
    print(account.role())
```

الناتج الفعلي

```
User
Admin
```

الكود اللي بينادي `role()` مش محتاج يعرف النوع الداخلي؛ يكفي إن الـObject بتدعم الواجهة المطلوبة.

الفرق اللي يهمك Polymorphism مرتبط بالواجهة والسلوك، ومش لازم يكون ناتج عن Inheritance فقط.

Polymorphism

Duck Typing

لو الـObject بتدعم Method المطلوبة، نقدر نستخدمها حتى من غير Parent مشتركة.

كائنات مستقلة بنفس الواجهة

PYTHON

```python
class Email:
    def send(self, text):
        return "Email: " + text

class Sms:
    def send(self, text):
        return "SMS: " + text

for channel in [Email(), Sms()]:
    print(channel.send("Hello"))
```

الناتج الفعلي

```
Email: Hello
SMS: Hello
```

Inheritance	Duck Typing
قد توجد Class أساسية مشتركة	لا يشترط Parent مشتركة
بتربط الأنواع بشكل صريح	بتعتمد على السلوك المتاح

انتبه Duck Typing مش معناها نتجاهل الأخطاء؛ اكتب عقودًا مفهومة للـMethods واختبر الـObjects المتوقعة.

Abstraction

Abstraction .07

بتحدد العمليات المطلوبة من غير ما تفرض تفاصيل التنفيذ.

Abstract Base Class

PYTHON

```python
from abc import ABC, abstractmethod

class Payment(ABC):
    @abstractmethod
    def pay(self, amount):
        pass

class Cash(Payment):
    def pay(self, amount):
        return f"Cash: {amount}"

print(Cash().pay(150))
```

الناتج الفعلي

```
Cash: 150
```

`ABC` مع `@abstractmethod` تمنع إنشاء نسخة من Class فيها Methods مجردة لم يتم تنفيذها. ده مفيد لما كل الأنواع لازم تلتزم بواجهة.

الفرق اللي يهمك Abstraction بتحدد ماذا يفعل الكائن؛ Encapsulation بتنظم كيفية الوصول لبياناته.

Abstraction

Abstract Methods في تطبيق واقعي

نفس العقد ينفع مع أكثر من استراتيجية دفع.

تنفيذان لنفس العقد

PYTHON

```python
from abc import ABC, abstractmethod

class Discount(ABC):
    @abstractmethod
    def apply(self, price):
        pass

class Fixed(Discount):
    def apply(self, price):
        return price - 20

class Percent(Discount):
    def apply(self, price):
        return price * 0.9

for item in [Fixed(), Percent()]:
    print(item.apply(200))
```

الناتج الفعلي

```
180
180.0
```

جرّب بنفسك اعمل Class جديدة NoDiscount ترجع السعر كما هو، ثم جرّبها في نفس الـLoop.

Special Methods

08. `__str__` و `__repr__`

تحكم في عرض الـObject كنص للمستخدم أو للمبرمج.

طريقة عرض Object

PYTHON

```python
class Course:
    def __init__(self, title):
        self.title = title

    def __str__(self):
        return self.title

    def __repr__(self):
        return f"Course({self.title!r})"

course = Course("Python")
print(str(course))
print(repr(course))
```

الناتج الفعلي

```
Python
Course('Python')
```

`__str__`	`__repr__`
تمثيل مقروء للمستخدم	تمثيل مفيد للمطور وأدوات Debugging

انتبه `__repr__` يفضّل يكون واضحًا وغير مضلل حتى لو مش Expression يمكن تشغيلها حرفيًا.

Special Methods

9 __len__ __eq__

Special Methods بتخلي الـObject تتعامل مع عمليات Python المألوفة.

طول مجموعة ومقارنة قيم

PYTHON

```python
class Team:
    def __init__(self, names):
        self.names = names

    def __len__(self):
        return len(self.names)

    def __eq__(self, other):
        if not isinstance(other, Team):
            return NotImplemented
        return self.names == other.names

first = Team(["Ali", "Mona"])
second = Team(["Ali", "Mona"])
print(len(first))
print(first == second)
```

الناتج الفعلي

```
2
True
```

الفرق اللي يهمك `__eq__` للمساواة في المحتوى، و `is` للمقارنة بين هوية الـObjects.

OOP Design

09. Composition: `has-a`

استخدم Object داخل Object بدل وراثة غير منطقية.

Course تحتوي Instructor

PYTHON

```python
class Instructor:
    def __init__(self, name):
        self.name = name

class Course:
    def __init__(self, title, instructor):
        self.title = title
        self.instructor = instructor

course = Course("Python", Instructor("Mona"))
print(course.title)
print(course.instructor.name)
```

الناتج الفعلي

```
Python
Mona
```

Inheritance: is-a	Composition: has-a
Student is a Person	Course has an Instructor
فيها إعادة استخدام عبر Parent	فيها تجميع Objects لها مسؤوليات مستقلة

جرّب بنفسك أضف Course تحتوي List من الطلاب، واطبع عددهم باستخدام `len()`.

OOP Design

`@dataclass` وتصميم نظيف

لما الـClass وظيفتها تخزين بيانات، Dataclass بتقلل الكود المتكرر.

Dataclass للبيانات

PYTHON

```python
from dataclasses import dataclass

@dataclass
class Product:
    name: str
    price: int

first = Product("Book", 80)
second = Product("Book", 80)
print(first)
print(first == second)
```

الناتج الفعلي

```
Product(name='Book', price=80)
True
```

استخدم	لما
`@dataclass`	معظم شغلها بيانات Entity
Class عادية	عندك سلوك وتحقيق شروط وتجميع مسؤوليات
`@property`	محتاج تحافظ على Validation أو حساب قيمة

انتبه Type Hints بتوضح النية، لكنها لا تفرض نوع القيمة تلقائيًا وقت التشغيل.

المشروع التطبيقي

10. مشروع: Course Manager

نظام بسيط للتسجيل والدرجات بتصميم OOP قابل للتوسعة.

Class	المسؤولية
`Person`	الاسم المشترك
`Student`	بيانات الطالب ودرجته
`Instructor`	اسم المدرس
`Course`	التسجيل والتحقق وحساب المتوسط

أنشئ ملف `course_manager.py`. الجزء التالي بداية الكود؛ كمّل باقي الملف من الصفحة التالية بالترتيب.

course_manager.py — الجزء 1/2

```python
class Person:
    def __init__(self, name):
        self.name = name

class Student(Person):
    def __init__(self, name):
        super().__init__(name)
        self.grade = None

class Instructor(Person):
    pass

class Course:
    def __init__(self, title, instructor, capacity=2):
        if capacity < 1:
            raise ValueError("Invalid capacity")
        self.title = title
        self.instructor = instructor
        self._capacity = capacity
        self.students = []

    @property
    def seats_left(self):
        return self._capacity - len(self.students)

    def register(self, student):
        if any(s.name == student.name for s in self.students):
            return "Duplicate"
        if self.seats_left == 0:
            return "Full"
        self.students.append(student)
```

انتبه `_capacity` خاصية داخلية، والـProperty `seats_left` بتقرأ المقاعد المتبقية من غير السماح بتعديلها مباشرة.

المشروع التطبيقي

المشروع: الكود الكامل (تكملة)

أضف الجزء الثاني مباشرة تحت الجزء الأول في نفس الملف.

course_manager.py — الجزء 2/2

```
        return "Added"

    def set_grade(self, name, grade):
        if not 0 <= grade <= 100:
            raise ValueError("Invalid grade")
        for student in self.students:
            if student.name == name:
                student.grade = grade
                return
        raise ValueError("Student not found")

    def average(self):
        grades = [s.grade for s in self.students if s.grade is not None]
        return sum(grades) / len(grades) if grades else None

if __name__ == "__main__":
    course = Course("Python Basics", Instructor("Mona"))
    print("Course:", course.title)
    print("Instructor:", course.instructor.name)
    print("Register Ali:", course.register(Student("Ali")))
    print("Register Sara:", course.register(Student("Sara")))
    print("Register Ali again:", course.register(Student("Ali")))
    print("Seats left:", course.seats_left)
    course.set_grade("Ali", 90)
    course.set_grade("Sara", 70)
    print("Grades: Ali=90, Sara=70")
    print("Average:", course.average())
    top = max(course.students, key=lambda s: s.grade)
    print(f"Top: {top.name} ({top.grade})")
    print("Third attempt:", course.register(Student("Omar")))
```

شغّل الملف باستخدام `python course_manager.py`. كل البيانات الموجودة في المثال ثابتة لضمان تطابق الناتج حرفيًا. وملف المصدر الكامل مرفق بشكل منفصل أيضًا.

جرّب بنفسك بعد التشغيل، غيّر `capacity=2` إلى 3 عند إنشاء Course، وجرّب تسجيل Omar قبل إدخال درجات الطلاب.

المشروع التطبيقي

المشروع: الناتج والتفسير

ماذا حدث عند التسجيل المكرر وامتلاء الكورس؟

الناتج الفعلي

```
Course: Python Basics
Instructor: Mona
Register Ali: Added
Register Sara: Added
Register Ali again: Duplicate
Seats left: 0
Grades: Ali=90, Sara=70
Average: 80.0
Top: Ali (90)
Third attempt: Full
```

السطر	التفسير
Register Ali: Added	أول تسجيل لاسم Ali
Register Sara: Added	المقعد الثاني اتشغل
Register Ali again: Duplicate	اسم الطالب موجود بالفعل
Seats left: 0	السعة 2 والتسجيلات 2
Average: 80.0	المتوسط = (90 + 70) / 2
Third attempt: Full	لا يوجد مقعد لطالب جديد

طوّر المشروع بنفسك

- أضف بحثًا عن طالب بالاسم مع نتيجة Not found.
- أضف Method تعرض الطلاب اللي حالتهم Passed فقط.
- أضف إمكانية تحديث الـCapacity مع شرط ألا تقل عن عدد المسجلين.

Debugging

الأخطاء الشائعة

اقرأ رسالة الخطأ واربطها بالسبب بدل الحفظ.

رسالة الخطأ	السبب المحتمل	تعمل إيه
`TypeError: ... takes ... arguments`	ناسي `self` أو Arguments غير مناسبة	راجع تعريف الـMethod وطريقة الاستدعاء
`AttributeError`	Attribute أو Method مش موجودة	راجع الاسم و `__init__`
`NameError`	اسم متغير أو Class غير معرّف	عرّفه قبل الاستخدام
`ValueError: Invalid grade`	التحقق رفض درجة خارج 0–100	افحص المدخلات قبل التخزين
`TypeError: Can't instantiate abstract class`	Abstract Method من غير تنفيذ	نفذ كل Methods المطلوبة
`RecursionError`	دالة بتستدعي نفسها من غير توقف	راجع Base Case
`KeyError`	الوصول لمفتاح غير موجود في Dictionary	استخدم `.get()` عند الحاجة
`IndentationError`	مسافات Block غلط	راجع إزاحة الأسطر

انتبه نص رسائل الأخطاء قد يختلف حسب إصدار Python والسياق؛ راجع نوع الاستثناء وسطره أولًا.

ورقة مراجعة

OOP Cheat Sheet

عايز أعمل كذا؟ قدامك أقصر صيغة تفتكرك.

عايز أعمل	الصيغة المختصرة
Class	`class Student:`
Object	`student = Student()`
Constructor	`def __init__(self, name):`
Instance Attribute	`self.name = name`
Instance Method	`def result(self):`
Inheritance	`class Student(Person):`
Parent Constructor	`super().__init__(name)`
Property	`@property`
Setter	`@grade.setter`
Class Method	`@classmethod` ثم `cls`
Static Method	`@staticmethod`
Abstract Class	`class Shape(ABC):`
Abstract Method	`@abstractmethod`
String Representation	`def __str__(self):`
Object Length	`def __len__(self):`
Dataclass	`@dataclass`
Composition	`self.instructor = instructor`
MRO	`MyClass.__mro__`

الفرق اللي يهمك متستخدمش Inheritance لمجرد مشاركة بيانات؛ اسأل هل العلاقة is-a أو has-a أولًا.

أسئلة

اختبر نفسك: توقّع الناتج

6 أسئلة قصيرة؛ اكتب الناتج الحرفي قبل صفحة الإجابات.

1. Class vs Instance — توقّع الناتج قبل ما تبص على الحل

```
class A:
    value = 2
x = A()
y = A()
x.value = 5
print(x.value, y.value)
```

2. Constructor — توقّع الناتج قبل ما تبص على الحل

```
class Item:
    def __init__(self, price=10):
        self.price = price
print(Item().price + Item(5).price)
```

3. Property — توقّع الناتج قبل ما تبص على الحل

```
class Box:
    @property
    def size(self):
        return 4
print(Box().size * 2)
```

4. Override — توقّع الناتج قبل ما تبص على الحل

```
class A:
    def label(self):
        return "A"
class B(A):
    def label(self):
        return super().label() + "B"
print(B().label())
```

5. Duck Typing — توقّع الناتج قبل ما تبص على الحل

```
class Cat:
    def sound(self):
        return "Meow"
class Dog:
    def sound(self):
        return "Woof"
print("-".join(x.sound() for x in [Cat(), Dog()]))
```

6. Dataclass — توقّع الناتج قبل ما تبص على الحل

```
from dataclasses import dataclass
@dataclass
class Point:
    x: int
    y: int
print(Point(1, 2) == Point(1, 2))
```

تمارين

اختبر نفسك: اكتب الكود

6 تمارين مستقلة؛ استخدم Classes وMethods بما يناسب المطلوب.

1. Student Label — اكتب Student فيها name وgrade وMethod اسمها label ترجع النص المطلوب.

```
Student("Mona",90).label() → Mona:90
```

2. Bank Account — اكتب BankAccount بمبلغ ابتدائي وMethod deposit ترفض القيم السالبة.

```
start=100, deposit(25) → Balance: 125
```

3. Inheritance — اعمل Employee وDeveloper يرث منها، وMethod role.

```
Developer("Nora").label() → Nora - Developer
```

4. Polymorphism — اعمل Email وSms لكل واحدة Method send(text).

```
send("Hi") → Email: Hi / SMS: Hi
```

5. Abstract Class — اعمل Shape مجردة وSquare تنفذ area().

```
Square(4).area() → 16
```

6. Composition — اعمل Item وCart يحتوي Items ويحسب مجموع qty × price.

```
[Item(20,2), Item(50,1)] → Total: 90
```

نصيحة استعمل أسماء المتغيرات اللي في السؤال، وخلي كل حل قابل للتشغيل لوحده.

الإجابات

الإجابات: توقع الناتج

كل ناتج متحقق منه بتشغيل البرنامج، والشرح يوضح سببه.

السؤال 1	`5 2`	التعديل على x أنشأ Instance Attribute تخص x، وy ظلت تقرأ من الـClass.
السؤال 2	`15`	الأول يأخذ Default=10 والثاني 5.
السؤال 3	`8`	الـProperty بتعرض قيمة 4؛ مضاعفتها 8.
السؤال 4	`AB`	`super()` ترجع A ثم تضيف B.
السؤال 5	`Meow-Woof`	الـObjects المستقلة تتبع نفس الواجهة sound().
السؤال 6	`True`	Dataclass بتنشيء مقارنة قيم للحقول افتراضيًا.

جرّب بنفسك غيّر قيمة واحدة في كل سؤال من الصفحة 29 واكتب كيف يتغير الناتج، ثم تحقق بالتشغيل.

الإجابات

حلول التمارين: 1-3

حل كامل مستقل تحت كل تمرين، والناتج هو ناتج التنفيذ.

تمرين 1 — Student Label

PYTHON

```python
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

    def label(self):
        return f"{self.name}:{self.grade}"

print(Student("Mona", 90).label())
```

الناتج الفعلي

```
Mona:90
```

تمرين 2 — Bank Account

PYTHON

```python
class BankAccount:
    def __init__(self, balance):
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount

account = BankAccount(100)
account.deposit(25)
print("Balance:", account.balance)
```

الناتج الفعلي

```
Balance: 125
```

تمرين 3 — Inheritance

PYTHON

```python
class Employee:
    def __init__(self, name):
        self.name = name

class Developer(Employee):
    def label(self):
        return self.name + " - Developer"

print(Developer("Nora").label())
```

الناتج الفعلي

```
Nora - Developer
```

الإجابات

حلول التمارين: 4–6

حل كامل مستقل تحت كل تمرين، والناتج هو ناتج التنفيذ.

تمرين 4 — Polymorphism

PYTHON

```python
class Email:
    def send(self, text):
        return "Email: " + text

class Sms:
    def send(self, text):
        return "SMS: " + text

for sender in [Email(), Sms()]:
    print(sender.send("Hi"))
```

الناتج الفعلي

```
Email: Hi
SMS: Hi
```

تمرين 5 — Abstract Class

PYTHON

```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Square(Shape):
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side ** 2

print(Square(4).area())
```

الناتج الفعلي

```
16
```

تمرين 6 — Composition

PYTHON

```python
class Item:
    def __init__(self, price, qty):
        self.price = price
        self.qty = qty

class Cart:
    def __init__(self, items):
        self.items = items

    def total(self):
        return sum(x.price * x.qty for x in self.items)

cart = Cart([Item(20, 2), Item(50, 1)])
print("Total:", cart.total())
```

الناتج الفعلي

```
Total: 90
```

الحالات الحدّية

Edge Cases & Test Table

الحالات الحدّية جزء من صحة التصميم مش خطوة إضافية.

edge_cases.py

```python
from course_manager import Course, Student, Instructor
c = Course("Python", Instructor("Mona"), capacity=1)
print("Empty average:", c.average())
print("Add:", c.register(Student("Ali")))
print("Duplicate:", c.register(Student("Ali")))
print("Full:", c.register(Student("Sara")))
for grade in [0, 100, -1, 101]:
    try:
        c.set_grade("Ali", grade)
        print("Accepted:", grade)
    except ValueError:
        print("Rejected:", grade)
```

الناتج الفعلي

```
Empty average: None
Add: Added
Duplicate: Duplicate
Full: Full
Accepted: 0
Accepted: 100
Rejected: -1
Rejected: 101
```

الحالة	النتيجة المتوقعة
قائمة بلا درجات	`Empty average: None`
طالب مكرر	`Duplicate`
السعة ممتلئة	`Full`
درجة = 0 أو 100	مقبولة
درجة = -1 أو 101	`Rejected`

انتبه الـaverage ترجع None لما مفيش درجات؛ ده أفضل من القسمة على صفر.

مراجع

المراجع والخطوة التالية

مصادر رسمية تراجع منها لما تحتار في سلوك Method أو Decorator.

Python Tutorial: Classes — Classes وInheritance وScope

https://docs.python.org/3.12/tutorial/classes.html

Python Data Model — Special Methods وسلوك Objects

https://docs.python.org/3.12/reference/datamodel.html

abc — Abstract Base Classes — ABC وabstractmethod

https://docs.python.org/3.12/library/abc.html

dataclasses — Dataclass والتوليد التلقائي

https://docs.python.org/3.12/library/dataclasses.html

Built-in Functions — isinstance وlen وغيرها

https://docs.python.org/3.12/library/functions.html

Exceptions — أنواع أخطاء Python

https://docs.python.org/3.12/library/exceptions.html

خطوة عملية بعد المذكرة

نفّذ Course Management System من غير نسخ. أضف Course ثانية، واعمل Function تعرض تقرير كل طالب، ثم اكتب 5 اختبارات لـCapacity وDuplicate وGrade Validation.

مراجعة النسخة المحتوى متوافق مع Python 3.12؛ الاختبارات نُفذت على Python 3.13.5 مع فحص Grammar 3.12. البيانات والنواتج موثقة بملف Verification JSON.