مذكرة مراجعة عملية

Python Fundamentals

أساسيات Python من الصفر حتى Functions

تفهم الفكرة، تقرأ الكود، تتوقع الناتج، وتختبر نفسك. مرجع مختصر للمذاكرة السريعة والتطبيق.

Python 3.12 | Variables → Recursion | 35 صفحة

إعداد

محمود منصور

Mahmoud Mansour

المنصة التعليمية

khatwa tech

youtube.com/@khatwatech

23 نوفمبر 2026

دليل الاستخدام

خريطة المذكرة

35 صفحة مركزة؛ كل مثال منفصل وقابل للتشغيل.

القسم	الصفحات
01. Introduction	3
02. Variables & Data Types	4–5
03. Input & Output	6
04. Operators & Casting	7–8
05. Strings	9–10
06. Conditions	11–12
07. Loops	13–15
08. Lists	16–17
09. Tuples & Sets	18
10. Dictionaries	19–20
11. Functions & Recursion	21–24
12. Project: Student Grades Manager	25–27
الأخطاء الشائعة + Cheat Sheet	28–29
اختبر نفسك: 6 + 6	30–31
الإجابات + Edge Cases	32–34
المراجع والخطوة التالية	35

تستخدم المذكرة إزاي؟

1. اقرأ ملخص الفكرة في كل قسم، وبعدها جرّب تتوقع الناتج قبل ما تبص تحته.
2. شغّل الأمثلة بنفسك، وغيّر قيمة واحدة في كل مرة واعمل Dry Run.
3. حل أسئلة الصفحتين 30–31 من غير الرجوع للإجابات، وبعدها راجع صفحات 32–34.

نطاق المذكرة الإصدار المستهدف Python 3.12. تشمل Recursion، ولا تشمل OOP أو List Comprehension.

البداية

01. Introduction

افهم شكل البرنامج وطريقة تشغيله قبل أول متغير.

Python لغة عالية المستوى بتنفّذ الأوامر باستخدام Interpreter. اكتب ملف بامتداد `.py`، وشغّله بالأمر `python file.py` أو `python3 file.py` حسب نظامك.

العنصر	فايدته
`print()`	عرض بيانات على الشاشة
`#`	تعليق لا يتم تنفيذه
Indentation	المسافات التي تحدد كتلة التعليمات
`.py`	امتداد ملف Python

أول برنامج

```python
# First Python program
print("Hello, Python!")
print(2 + 3)
```

الناتج الفعلي

```
Hello, Python!
5
```

Python بتنّفذ السطور بالترتيب من فوق لتحت.

المسافات داخل Block

```python
course = "Python"
if course == "Python":
    print("Ready")
```

الناتج الفعلي

```
Ready
```

خطأ مشهور كتابة السطر بعد `if` من غير مسافات بادئة يسبب `IndentationError`.

جرّب بنفسك غيّر `course` إلى `Java`، وتوقع هيظهر إيه.

المتغيرات · 1 / 2

02. Variables & Data Types

المتغير بيسمي البيانات؛ والـType بيحدد التعامل معاها.

الـVariable اسم مرتبط بقيمة، ومش محتاج تكتب نوعها وقت الإنشاء. النوع بيتحدد من القيمة نفسها.

Variables + type

```
student = "Mona"
age = 20
gpa = 3.5
is_active = True
print(student, age)
print(type(gpa).__name__)
print(type(is_active).__name__)
```

الناتج الفعلي

```
Mona 20
float
bool
```

Type	مثال	الوصف
`int`	`20`	عدد صحيح
`float`	`3.5`	عدد عشري
`str`	`"Mona"`	نص
`bool`	`True`	منطقي
`NoneType`	`None`	غياب قيمة

انتبه `True` و `False` بحروف كبيرة؛ و `"20"` نص مش رقم.

المتغيرات · 2 / 2

Variables & Data Types .02

اختار أسماء واضحة؛ وافتكر إن النوع ممكن يتغير مع الإسناد.

الـVariable ممكن تتغير قيمته، لكن الاسم لازم يلتزم بقواعد Python: يبدأ بحرف أو `_`، ومينفعش يبدأ برقم، وحسّاس لحالة الحروف.

إعادة الإسناد وNone

```
price = 100
price = 120
student_name = "Omar"
missing = None
print(price)
print(student_name)
print(missing)
```

الناتج الفعلي

```
120
Omar
None
```

Multiple Assignment وSwap

```
a, b = 4, 9
a, b = b, a
print(a, b)
print(isinstance(a, int))
```

الناتج الفعلي

```
9 4
True
```

صحيح	غير صحيح
`student_name`	`student-name`
`_count`	`2count`
`grade1`	`for`

الفرق اللي يهمك = معناها Assignment، أما == فهي مقارنة مساواة.

المدخلات والمخرجات

03. Input & Output

اقرأ البيانات، حوّل النوع الصحيح، واطبع نتيجة واضحة.

`print()` لعرض القيم، و`input()` لقراءة نص من المستخدم؛ حتى لو كتب رقم، الناتج الأساسي من `input()` نوعه `str`.

طرق الطباعة

```
course = "Python"
lessons = 8
print("Course:", course)
print(f"Lessons: {lessons}")
print("A", "B", sep=" | ")
print("Done", end="!")
```

الناتج الفعلي

```
Course: Python
Lessons: 8
A | B
Done!
```

قراءة الاسم والعمر

```
name = input()
age = int(input())
print(f"{name} is {age}")
```

الإدخال (Input): `'Mona\n19'`

الناتج الفعلي

```
Mona is 19
```

الأداة	الغرض
`print(x)`	طباعة القيمة
`print(a, b, sep="-")`	تغيير الفاصل
`print(x, end="!")`	تغيير نهاية السطر
`input()`	قراءة `str`
`int(input())`	تحويل الإدخال إلى عدد صحيح

خطأ مشهور `int(input())` يفشل بـ `ValueError` لو المستخدم كتب نصًا لا يمثل عددًا صحيحًا.

العمليات · 1 / 2

04. Operators & Casting

تعرف تحسب وتعرف الفرق بين ناتج القسمة وباقيها.

الـOperators بتنفذ عمليات حسابية أو مقارنات. فرّق خصوصًا بين `/` و `//` و `%`.

Arithmetic Operators

```
a, b = 17, 5
print(a + b)
print(a / b)
print(a // b)
print(a % b)
print(a ** 2)
```

الناتج الفعلي

```
22
3.4
3
2
289
```

Operator	المعنى	مثال
`+` , `-` , `*`	جمع وطرح وضرب	`12` → `4 * 3`
`/`	قسمة عادية	`2.5` → `5 / 2`
`//`	Floor Division	`2` → `5 // 2`
`%`	باقي القسمة	`1` → `5 % 2`
`**`	أس	`8` → `2 ** 3`

انتبه `//` بتقرب لأسفل، مش تجاه الصفر. مثلًا `-5 // 2` يساوي `-3`.

العمليات · 2 / 2

04. Operators & Casting

حوّل النوع قبل العمليات؛ واستخدم المقارنة والمنطق للشروط.

المقارنات بترجع `True` أو `False`. والتحويل بيغيّر نوع القيمة، لكنه مش بالضرورة يغيّر المتغير القديم.

Comparison & Logical

```
grade = 75
print(grade >= 50)
print(grade > 90 or grade == 75)
print(not (grade < 50))
print(50 <= grade <= 100)
```

الناتج الفعلي

```
True
True
True
True
```

Type Casting

```
text = "24"
number = int(text)
price = float("12.5")
print(number + 6)
print(price)
print(str(number) + " years")
```

الناتج الفعلي

```
30
12.5
24 years
```

Operator	التفسير
`and`	كل الشروط صحيحة
`or`	شرط واحد على الأقل صحيح
`not`	عكس القيمة المنطقية
`+=`	تحديث المتغير باستخدام جمع

الفرق اللي يهمك `int(3.9)` نتيجته `3` ؛ بيحذف الجزء العشري تجاه الصفر، ومش بيقرب لأقرب عدد.

النصوص · 1 / 2

05. Strings

اتعامل مع النص كمجموعة أحرف لها Index وSlice.

`str` سلسلة أحرف غير قابلة للتعديل مباشرة (immutable). أول حرف Index بتاعه `0` وآخر حرف ممكن توصله بـ `-1` .

Indexing & Slicing

```
course = "Python"
print(course[0])
print(course[-1])
print(course[1:4])
print(course[:2])
print(len(course))
```

الناتج الفعلي

```
P
n
yth
Py
6
```

String Methods

```
name = "  Mona Ali  "
print(name.strip())
print(name.strip().upper())
print("Py" in "Python")
```

الناتج الفعلي

```
Mona Ali
MONA ALI
True
```

خطأ مشهور `course[0] = "J"` مش مسموح؛ أنشئ String جديدة بدل تعديل الحرف في مكانه.

النصوص · 2 / 2

05. Strings

قسّم النصوص ونسّقها، وافصل بين القيمة الأصلية والناتج الجديد.

دوال الـString غالبًا بترجع نصًا جديدًا. استخدم `split()` للتقسيم و`join()` للدمج، وf-strings لعرض القيم بطريقة مقروءة.

split / join / replace

```
courses = "Python,Java,C"
parts = courses.split(",")
print(parts)
print(" | ".join(parts))
print("banana".replace("a", "o"))
```

الناتج الفعلي

```
['Python', 'Java', 'C']
Python | Java | C
bonono
```

Escape Characters وf-string

```
student = "Omar"
grade = 87.456
print(f"{student}: {grade:.1f}")
print("Line1\nLine2")
```

الناتج الفعلي

```
Omar: 87.5
Line1
Line2
```

Method	الفايدة
`upper()` / `lower()`	تغيير حالة الحروف
`strip()`	حذف المسافات من الأطراف
`find("x")`	أول Index أو `-1`
`count("a")`	عدّ التكرارات
`startswith("Py")`	اختبار بداية النص

الشروط · 1 / 2

06. Conditions

اتخذ قرار بناءً على شرط واحد أو أكتر.

`if` بتنّفذ كتلة لو الشرط صحيح، و`elif` بتختبر بدائل بالترتيب، و`else` لأي حالة متبقية.

if / elif / else

```
grade = 76
if grade >= 90:
    print("Excellent")
elif grade >= 50:
    print("Passed")
else:
    print("Failed")
```

الناتج الفعلي

```
Passed
```

Conditional Expression

```
total = 220
status = "Free shipping" if total >= 200 else "Shipping fee"
print(status)
```

الناتج الفعلي

```
Free shipping
```

انتبه الترتيب مهم: شرط `grade >= 90` لازم يسبق `grade >= 50` في المثال.

الشروط · 2 / 2

06. Conditions

افهم Truthy وFalsy، وخلي الشروط المتداخلة واضحة.

Python بتعتبر بعض القيم False في السياق المنطقي، زي `0` ، و `""` ، و `[]` ، و `None` . مع إن كل قيمة ليها Type مختلف.

Truthy / Falsy

```
names = []
if not names:
    print("No students")
else:
    print("Has students")
print(bool("0"))
print(bool(0))
```

الناتج الفعلي

```
No students
True
False
```

Nested if

```
age = 19
has_ticket = True
if age >= 18:
    if has_ticket:
        print("Allowed")
    else:
        print("Need ticket")
else:
    print("Too young")
```

الناتج الفعلي

```
Allowed
```

قيمة	bool(...)
`0` ,`0.0`	`False`
`""` ,`[]` ,`{}`	`False`
`None`	`False`
`"0"` ,`[0]`	`True`

التكرار · 1 / 3

07. Loops

كرر على عناصر أو عدد مرات معلوم باستخدام for.

`for` بتعدّي على عناصر Sequence أو Range. و `range(start, stop, step)` لا تشمل `stop`.

for + range

```
for number in range(1, 5):
    print(number, end=" ")
print()
```

الناتج الفعلي

```
1 2 3 4
```

Loop على List

```
courses = ["Python", "Java", "C"]
for course in courses:
    print(course)
```

الناتج الفعلي

```
Python
Java
C
```

الفرق اللي يهمك `range(3)` بتطلع 0 و1 و2، مش 3.

التكرار · 2 / 3

07. Loops

while للشرط المستمر؛ وAccumulator لتجميع النتائج.

`while` بتكرر طول ما الشرط صحيح. لازم قيمة تتغير أو يبقى فيه طريق للخروج؛ وإلا يحصل Infinite Loop.

while Loop

```
count = 3
while count > 0:
    print(count)
    count -= 1
print("Done")
```

الناتج الفعلي

```
3
2
1
Done
```

Accumulator + Average

```
total = 0
for grade in [40, 60, 80]:
    total += grade
print(total)
print(total / 3)
```

الناتج الفعلي

```
180
60.0
```

خطأ مشهور متنساش تحديث العداد جوا `while` . ولو بتقسّم على عدد عناصر، اتأكد إنه مش صفر.

التكرار · 3 / 3

07. Loops

تحكم في التكرار وابنِ Loop متداخلة بأقل تعقيد ممكن.

`break` بتوقف الـLoop بالكامل، أما `continue` فبتتخطّى باقي الدورة الحالية.

break / continue

```
for value in range(1, 7):
    if value == 3:
        continue
    if value == 6:
        break
    print(value, end=" ")
print()
```

الناتج الفعلي

```
1 2 4 5
```

Nested Loops

```
for row in range(1, 3):
    for col in range(1, 4):
        print(f"{row}-{col}", end=" ")
    print()
```

الناتج الفعلي

```
1-1 1-2 1-3
2-1 2-2 2-3
```

الكلمة	التأثير
`break`	إنهاء أقرب Loop
`continue`	تجاهل بقية الدورة
`pass`	تعليمة لا تعمل شيئًا

القوائم · 1 / 2

08. Lists

List مناسبة لتخزين مجموعة عناصر بترتيب قابل للتغيير.

الـList بتجمع عناصر قابلة للتعديل، ممكن تضيف وتحذف وترتب. أول Index هو 0، و `len()` بترجع عدد العناصر.

إنشاء وتعديل List

```
grades = [80, 55, 90]
grades.append(70)
grades[1] = 60
print(grades)
print(grades[0])
print(len(grades))
```

الناتج الفعلي

```
[80, 60, 90, 70]
80
4
```

Indexing / Slicing

```
courses = ["C", "Java", "Python"]
print(courses[1:])
print(courses[-1])
print("Python" in courses)
```

الناتج الفعلي

```
['Java', 'Python']
Python
True
```

انتبه الوصول إلى Index خارج حدود الـList بيعمل `IndexError`.

القوائم · 2 / 2

08. Lists

اعرف إمتى تعدّل الأصل وإمتى تحتاج قائمة جديدة.

بعض الـMethods بتعدّل الـList نفسها وترجع `None` ؛ مثال: `sort()` . لو عايز نسخة مرتبة من غير تعديل الأصل استخدم `sorted()` .

ترتيب وحساب

```
grades = [70, 40, 90, 60]
grades.sort()
print(grades)
print(sum(grades))
print(max(grades))
print(min(grades))
```

الناتج الفعلي

```
[40, 60, 70, 90]
260
90
40
```

إضافة وحذف

```
names = ["Mona", "Ali", "Omar"]
names.remove("Ali")
names.insert(1, "Sara")
print(names)
print(names.pop())
print(names)
```

الناتج الفعلي

```
['Mona', 'Sara', 'Omar']
Omar
['Mona', 'Sara']
```

الأمر	النتيجة
`append(x)`	إضافة في الآخر
`insert(i, x)`	إضافة عند Index
`remove(x)`	حذف أول عنصر يساوي x
`pop()`	حذف وإرجاع آخر عنصر
`sorted(items)`	إرجاع قائمة مرتبة جديدة

المجموعات

09. Tuples & Sets

اختار النوع حسب الاحتياج للترتيب والتكرار والتعديل.

الـTuple عناصرها مرتبة وغير قابلة للتعديل؛ والـSet بتحتفظ بقيم فريدة من غير ترتيب تعتمد عليه. بنستخدم `sorted()` عشان نعرض الـSet بنتيجة ثابتة.

Tuple

```
course = ("Python", 12)
print(course[0])
print(len(course))
print(course[1])
```

الناتج الفعلي

```
Python
2
12
```

Set وعملياتها

```
a = {1, 2, 2, 3}
b = {3, 4}
print(sorted(a))
print(sorted(a | b))
print(sorted(a & b))
```

الناتج الفعلي

```
[1, 2, 3]
[1, 2, 3, 4]
[3]
```

النوع	مرتب؟	يقبل التكرار؟	قابل للتعديل؟
List	نعم	نعم	نعم
Tuple	نعم	نعم	لا
Set	لا تعتمد على ترتيب	لا	نعم

القواميس · 1 / 2

10. Dictionaries

مفتاح واضح بيخلي الوصول للبيانات أسهل من البحث بالموقع.

الـDictionary بتربط Key بقيمة. الوصول بالقوسين لKey غير موجودة يعمل `KeyError`؛ واستخدم `get()` لو محتاج قيمة افتراضية.

إنشاء وتحديث Dict

```
student = {"name": "Mona", "grade": 87}
print(student["name"])
print(student.get("city", "Unknown"))
student["grade"] = 90
print(student["grade"])
```

الناتج الفعلي

```
Mona
Unknown
90
```

()items

```
prices = {"Book": 100, "Pen": 15}
for item, price in prices.items():
    print(f"{item}: {price}")
```

الناتج الفعلي

```
Book: 100
Pen: 15
```

انتبه الـKeys لازم تكون Hashable؛ مثلًا `str` و `int` و `tuple` المناسبة، لكن `list` مش Key صالحة.

القواميس · 2 / 2

10. Dictionaries

استخدم Dictionary عندما تكون البيانات لها أسماء مفاتيح.

القواميس مفيدة لدرجات الطلاب والأسعار. استخدم `in` لاختبار وجود Key، و `values()` لجمع القيم.

Keys & Values

```
grades = {"Mona": 90, "Ali": 70}
print("Mona" in grades)
print(list(grades.keys()))
print(sum(grades.values()))
print(len(grades))
```

الناتج الفعلي

```
True
['Mona', 'Ali']
160
2
```

إضافة وحذف

```
cart = {"Book": 80}
cart["Pen"] = 10
removed = cart.pop("Book")
print(removed)
print(cart)
```

الناتج الفعلي

```
80
{'Pen': 10}
```

Method	الغرض
`get(key, default)`	قراءة آمنة
`keys()`	المفاتيح
`values()`	القيم
`items()`	أزواج Key/Value
`pop(key)`	حذف وإرجاع القيمة

الدوال · 1 / 4

11. Functions

قسم المشكلة لدوال صغيرة، كل واحدة تعمل مهمة واضحة.

الـFunction كتلة قابلة لإعادة الاستخدام. بنعرفها بـ `def` ، وبنشغّلها بكتابة اسمها وأقواس الاستدعاء.

تعريف واستدعاء

```
def greet(name):
    print(f"Hello, {name}")

greet("Mona")
greet("Omar")
```

الناتج الفعلي

```
Hello, Mona
Hello, Omar
```

Parameters & Arguments

```
def course_info(course, lessons):
    print(f"{course}: {lessons} lessons")

course_info("Python", 12)
```

الناتج الفعلي

```
Python: 12 lessons
```

فرق مهم Parameter الاسم في تعريف الـFunction؛ Argument القيمة التي تمررها وقت الاستدعاء.

الدوال · 2 / 4

11. Functions

الدالة اللي بترجع قيمة أسهل في الاختبار وإعادة الاستخدام.

`return` بترجع قيمة للبرنامج المستدعي، بينما `print()` بتعرض قيمة للمستخدم. لو الدالة خلصت من غير `return`، بترجع `None`.

Return

```
def add(a, b):
    return a + b

result = add(7, 5)
print(result)
print(add(10, 2))
```

الناتج الفعلي

```
12
12
```

Default & Keyword Arguments

```
def discount(price, rate=10):
    return price * (1 - rate / 100)

print(discount(200))
print(discount(price=200, rate=25))
```

الناتج الفعلي

```
180.0
150.0
```

انتبه متستخدمش List كـMutable Default Argument في الأساسيات؛ استخدم `None` وعرّف List داخل الدالة عند الحاجة.

الدوال · 3 / 4

11. Functions

افصل نطاق المتغيرات وغطّي الحالات غير المعتادة قبل الحساب.

الـScope بتحدد أين يمكن استخدام الاسم. المتغير الذي تنشئه داخل Function محلي افتراضيًا. ممكن ترجع القيمة بدل تعديل Global.

Local vs Global

```
grade = 80

def update_grade(value):
    grade = value + 5
    return grade

print(update_grade(80))
print(grade)
```

الناتج الفعلي

```
85
80
```

Guard Clause للقائمة الفاضية

```
def calculate(grades):
    if not grades:
        return 0
    return sum(grades) / len(grades)

print(calculate([50, 70, 90]))
print(calculate([]))
```

الناتج الفعلي

```
70.0
0
```

الفرق اللي يهمك القائمة Mutable؛ لو عدّلت عناصرها داخل الدالة هيظهر التغيير خارجها. ده مختلف عن إعادة إسناد متغير محلي.

الدوال · 4 / 4

11. Functions & Recursion

قسم الحل لجزء أصغر وحدد شرط توقف واضح.

Recursion هي إن الدالة تنادي نفسها لحل جزء أصغر من المشكلة. لازم Base Case يوقف الاستدعاءات.

Recursion: factorial

```
def factorial(n):
    if n <= 1:
        return 1
    return n * factorial(n - 1)

print(factorial(5))
print(factorial(0))
```

الناتج الفعلي

```
120
1
```

Base Case مع طباعة

```
def countdown(n):
    if n == 0:
        print("Go!")
        return
    print(n)
    countdown(n - 1)

countdown(3)
```

الناتج الفعلي

```
3
2
1
Go!
```

انتبه

الـfactorial في المثال يفترض أن `n` عدد صحيح غير سالب. إهمال Base Case قد يسبب `RecursionError`. استخدم Loop غالبًا لو المسألة تكرار مباشر وكبير.

المشروع · 1 / 3

12. Project — Student Grades Manager

برنامج Console يجمع Input وLists وDictionaries وFunctions.

المطلوب: اقرأ عدد الطلاب وأسماءهم ودرجاتهم، ارفض العدد السالب أو اسمًا فارغًا أو درجة خارج 0–100، وبعدها اطبع حالة كل طالب وتقرير المجموعة.

Input → Process → Output

المرحلة	التفاصيل
Input	عدد الطلاب ثم الاسم والدرجة لكل طالب
Validation	رفض الاسم الفارغ، والدرجة خارج `[0, 100]`، والأعداد غير الصالحة
Process	تجميع البيانات في `list` من `dict`؛ حساب المتوسط وأعلى/أقل درجة
Output	`Passed` / `Failed` لكل طالب ثم ملخص رقمي

الدوال المستخدمة

الدالة	دورها
`read_grades()`	تقرأ وتتحقق من البيانات
`analyze(students)`	تحسب الإحصائيات وتتجنب القسمة على صفر
`main()`	ينظم سير التنفيذ ويعرض التقرير

قاعدة النجاح الطالب `Passed` إذا كانت درجته ≥ 50، وإلا `Failed`. الدرجة 50 ناجح بالضبط.

كيف تجرّب المشروع؟ احفظ الكود التالي في ملف `grades.py` ثم شغّله؛ اكتب كل قيمة Input في سطر مستقل.

المشروع · 2 / 3

12. Project — الكود الكامل

أنشئ ملف grades.py وانسخ الكود التالي بالكامل.

الكود مستقل، ويستخدم Functions وList وDictionary و `try / except` لضبط الإدخال، من غير OOP أو List Comprehension.

grades.py — الكود الكامل (شغّله كما هو)

```python
def read_grades():
    count = int(input())
    if count < 0:
        print("Invalid count")
        return None
    students = []
    for _ in range(count):
        name = input().strip()
        grade = int(input())
        if not name or not 0 <= grade <= 100:
            print("Invalid student")
            return None
        students.append({"name": name, "grade": grade})
    return students


def analyze(students):
    if not students:
        return {"average": 0, "highest": None,
                "lowest": None, "passed": 0, "failed": 0}
    grades = []
    for student in students:
        grades.append(student["grade"])
    passed = 0
    failed = 0
    for grade in grades:
        if grade >= 50:
            passed += 1
        else:
            failed += 1
    return {
        "average": sum(grades) / len(grades),
        "highest": max(grades),
        "lowest": min(grades),
        "passed": passed,
        "failed": failed,
    }


def main():
    try:
        students = read_grades()
        if students is None:
            return
        report = analyze(students)
        for student in students:
            status = "Passed" if student["grade"] >= 50 else "Failed"
            print(f"{student['name']}: {status}")
        print(f"Average: {report['average']:.1f}")
        print(f"Highest: {report['highest']}")
        print(f"Lowest: {report['lowest']}")
        print(f"Passed: {report['passed']}")
        print(f"Failed: {report['failed']}")
    except ValueError:
        print("Invalid number")


if __name__ == "__main__":
    main()
```

المشروع · 3 / 3

12. Project — النتيجة والتطوير

قارن المخرجات بالنص كما هي، ثم طوّر وظيفة واحدة.

Input حرفيًا (كل قيمة في سطر):

```
3
Mona
90
Ali
40
Omar
70
```

الناتج الفعلي للمشروع

```
Mona: Passed
Ali: Failed
Omar: Passed
Average: 66.7
Highest: 90
Lowest: 40
Passed: 2
Failed: 1
```

السطر / القيمة	التفسير
`Mona: Passed`	90 ≥ 50
`Ali: Failed`	40 أصغر من 50
`Omar: Passed`	70 ≥ 50
`Average: 66.7`	`(90 + 40 + 70) / 3` بعد تنسيق منزلة عشرية
`Lowest: 40` / `Highest: 90`	أكبر وأصغر درجة
`Failed: 1` / `Passed: 2`	عدد الناجحين والراسبين

حالات مهمّة للمشروع

Input مختصر	الناتج الأساسي
`0` طلاب	`Average: 0.0` و `Highest: None`
طالب بدرجة `50`	`Passed: 1`
طالب بدرجة `101`	`Invalid student`
درجة غير رقمية	`Invalid number`

3 أفكار لتطوير المشروع

1. اعرض اسم صاحب أعلى درجة، مش الدرجة فقط.
2. خلّي المستخدم يدخل عدة مجموعات في تشغيل واحد.
3. اطبع تقدير A / B / C / F حسب مدى الدرجة.

تشخيص سريع

الأخطاء الشائعة

افهم نوع الخطأ بدل حفظ رسالة طويلة قد تختلف حسب البيئة.

الرسالة بتقول لك نوع المشكلة، لكن لازم تقرأ السطر المسبب وتراجع القيمة أو الـIndex أو الـIndentation.

رسالة الخطأ	السبب المحتمل	تعمل إيه؟
`SyntaxError`	صياغة ناقصة أو رمز غلط	راجع `:` والأقواس والاقتباس
`IndentationError`	المسافات داخل Block مش صحيحة	استخدم 4 مسافات ثابتة
`NameError`	اسم متغير غير معرّف	تأكد من الاسم ومكان تعريفه
`TypeError`	عملية بين أنواع غير مناسبة	حوّل النوع أو صحّح العملية
`ValueError`	تحويل قيمة غير صالحة	تحقق من النص قبل `int()`
`IndexError`	Index خارج الـList	راجع `len()` ومجال الفهرس
`KeyError`	Key غير موجودة	استخدم `get()` أو افحص `in`
`ZeroDivisionError`	المقام صفر	افحص العدد قبل القسمة
`RecursionError`	استدعاءات Recursion بلا توقف	راجع الـBase Case

انتبه مثال فشل: `int("abc")` يسبب `ValueError` ، مش `TypeError` .

ملخص في صفحة

ورقة مراجعة سريعة — Cheat Sheet

أهم الصيغ لحد Functions بدون إضافات متقدمة.

عايز أعمل كذا	الصيغة المختصرة
أطبع نص	`print("Hello")`
أقرأ نص	`name = input()`
أقرأ رقم صحيح	`age = int(input())`
أعرف النوع	`type(value)`
أكتب شرط	`if grade >= 50:`
أكتب فرع تاني	`elif grade >= 90:`
أكتب بديل	`else:`
أكرر 5 مرات	`for i in range(5):`
أكرر بشرط	`while count > 0:`
أضيف للقائمة	`items.append(x)`
أجيب عدد العناصر	`len(items)`
أجمع أرقام	`sum(grades)`
أرتّب من غير تعديل الأصل	`sorted(items)`
أقرأ من Dict بأمان	`student.get("grade", 0)`
أعرّف Function	`def add(a, b):`
أرجع قيمة	`return a + b`
أحمي القسمة من صفر	`if not grades: return 0`
أكتب Recursion	`if n <= 1: return 1`
أحوّل String لعدد	`int(text)`
أكتب f-string	`f"Grade: {grade}"`

مراجعة في دقيقة اسأل نفسك قبل أي مسألة: المدخلات؟ المطلوب؟ النوع؟ الحالات الحدّية؟

6 أسئلة

اختبر نفسك — توقّع الناتج

حل بدون تشغيل؛ ثم قارن النتائج صفحة 32.

قبل ما تفتح صفحة الإجابات، اكتب الناتج حرفيًا، بما فيه ترتيب السطور والمسافات.

Q1. توقّع الناتج

توقّع الناتج بعد تحديث المتغير.

```
x = 7
x += 3
print(x * 2)
```

Q2. توقّع الناتج

فرّق بين Floor Division وRemainder.

```
print(17 // 5)
print(17 % 5)
```

Q3. توقّع الناتج

افتكر إن stop غير شامل.

```
word = "Python"
print(word[1:4])
```

Q4. توقّع الناتج

راجع التعديل وطول القائمة.

```
numbers = [2, 4]
numbers.append(6)
print(numbers[-1])
print(len(numbers))
```

Q5. توقّع الناتج

خد بالك من > بدل >=.

```
grade = 50
if grade > 50:
    print("High")
else:
    print("At most 50")
```

Q6. توقّع الناتج

اعمل Trace لأول 3 استدعاءات.

```
def f(n):
    if n == 0:
        return 0
    return n + f(n - 1)
print(f(3))
```

6 تمارين

اختبر نفسك — اكتب الحل بإيدك

الهدف إنك تكتب Function بنفسك وتختبرها بأكثر من حالة.

اكتب حل كامل لكل تمرين من غير List Comprehension. أمثلة Input → Output مرجعية للاختبار، ومطلوب أيضًا التفكير في القائمة الفاضية والحدود.

E1. اطبع مجموع السعرين بعد خصم ثابت.

→ `price1=80, price2=45, discount=10`

`115`

E2. اكتب Function تعيد Passed أو Failed حسب الدرجة.

`Passed` → `grade=50`

E3. اجمع الأرقام الزوجية من 1 إلى n.

`12` → `n=6`

E4. أرجع أكبر قيمة من List، وللفاضية اطبع Empty.

`90` → `grades=[70, 90, 40]`

E5. احسب متوسط قيم Dictionary، وللفاضية اطبع 0.

→ `grades={"Mona":80,"Ali":60}`

`70.0`

E6. اكتب factorial بـRecursion؛ 0! = 1.

`120` → `n=5`

جرّب بنفسك اختبر حلك على 0 وعلى الحد 50 وعلى List فاضية، حتى لو المثال الأساسي ما استخدمش الحالات دي.

الحلول · 1 / 3

الإجابات — توقّع الناتج

6 نواتج مفسّرة خطوة بخطوة.

كل ناتج اتولد بالتشغيل. التفسير يوضح الفكرة التي أدت للنتيجة، مش مجرد نقل الناتج.

Q1 — الناتج

```
20
```

`x` بقت 10، وبعد الضرب في 2 الناتج 20.

Q2 — الناتج

```
3
2
```

`//` ترجع 3، و`%` ترجع باقي القسمة 2.

Q3 — الناتج

```
yth
```

العناصر عند Index 1 و2 و3 هي `yth` .

Q4 — الناتج

```
6
3
```

بعد `append` آخر عنصر 6، وعدد العناصر 3.

Q5 — الناتج

```
At most 50
```

الدرجة تساوي 50، فالشرط `> 50` خطأ.

Q6 — الناتج

```
6
```

`f(3)` تساوي 3 + 2 + 1 + 0 = 6.

الحلول · 2 / 3

الإجابات — تمارين E1 إلى E3

الحل + الناتج + الفكرة وراءه.

كل تمرين له حل كامل قابل للتشغيل، مع الناتج الحرفي.

الحل E1

```
price1 = 80
price2 = 45
discount = 10
print(price1 + price2 - discount)
```

الناتج الفعلي

```
115
```

اجمع الأول ثم اطرح الخصم مرة واحدة.

الحل E2

```
def status(grade):
    return "Passed" if grade >= 50 else "Failed"

print(status(50))
```

الناتج الفعلي

```
Passed
```

الحد 50 ضمن النجاح.

الحل E3

```
def even_sum(n):
    total = 0
    for value in range(1, n + 1):
        if value % 2 == 0:
            total += value
    return total

print(even_sum(6))
```

الناتج الفعلي

```
12
```

الأرقام الزوجية: 2 + 4 + 6 = 12.

الحلول · 3 / 3

الإجابات — تمارين E4 إلى E6

تأكد من القيم الحدّية قبل اعتماد أي حل.

الحل E4

```
def highest(grades):
    if not grades:
        return "Empty"
    return max(grades)

print(highest([70, 90, 40]))
```

الناتج الفعلي

```
90
```

قبل `max()` تأكد إن القائمة غير فاضية.

الحل E5

```
def average(grades):
    if not grades:
        return 0
    return sum(grades.values()) / len(grades)

print(average({"Mona": 80, "Ali": 60}))
```

الناتج الفعلي

```
70.0
```

`values()` هي الدرجات؛ و `len()` هو عدد الطلاب.

الحل E6

```
def factorial(n):
    if n <= 1:
        return 1
    return n * factorial(n - 1)

print(factorial(5))
```

الناتج الفعلي

```
120
```

قاعدة التوقف `n <= 1` تعطي 1.

اختبارات الحالات الحدّية

الحالة	التجربة	الناتج
قائمة درجات فاضية	`highest([])`	`Empty`
Dictionary فاضي	`average({})`	`0`
على حد النجاح	`status(50)`	`Passed`
قبل حد النجاح	`status(49)`	`Failed`
أقل قيمة factorial	`factorial(0)`	`1`
جمع زوجي بلا عناصر	`even_sum(0)`	`0`

قراءة إضافية

المراجع والخطوة التالية

مصادر رسمية وخطة انتقال عملية بعد إنهاء الأساسيات.

المرجع الأساسي للمذكرة هو توثيق Python 3.12 الرسمي. الروابط التالية تثبّت القواعد المستخدمة في كل موضوع.

Python 3.12 — Tutorial
أساسيات Python وتجربة أول برنامج
https://docs.python.org/3.12/tutorial/

An Informal Introduction
Data Types وStrings وLists
https://docs.python.org/3.12/tutorial/introduction.html

Control Flow & Functions
Conditions وLoops وFunctions
https://docs.python.org/3.12/tutorial/controlflow.html

Data Structures
Lists وDictionaries وTuples وSets
https://docs.python.org/3.12/tutorial/datastructures.html

Built-in Types
Strings وعمليات الأنواع
https://docs.python.org/3.12/library/stdtypes.html

Built-in Functions
print وinput وlen وsum وmax
https://docs.python.org/3.12/library/functions.html

Exceptions
أسباب الأخطاء وتصنيفها
https://docs.python.org/3.12/library/exceptions.html

Errors and Exceptions
قراءة الأخطاء والتعامل معها
https://docs.python.org/3.12/tutorial/errors.html

خطوتك العملية بعد المذكرة

أعد كتابة Student Grades Manager من غير نسخ، واختبر: 0 طلاب، طالب بدرجة 50، درجة 100، قيمة غير رقمية. بعد كده انتقل لـFile Handling وModules وOOP.

ملاحظة التحقق الكود مكتوب وفق Syntax مدعوم في Python 3.12. تم تشغيل الأمثلة في بيئة Python 3.13 مع فحص Syntax مقابل قواعد 3.12؛ نتيجة البيئة موثّقة في سجل الاختبارات.