ملخص شامل للغة برمجة Python (حتى 8 أبريل 2025)

1. مقدمة عن لغة Python

ما هي لغة Python؟

Python هي لغة برمجة عالية المستوى، مُفسَّرة (interpreted)، تفاعلية (interactive)، وشيئية التوجه (object-oriented). تُعرف ببنيتها النحوية الواضحة والمقروءة التي تشبه اللغة الإنجليزية، مما يجعلها سهلة التعلم والاستخدام. صُممت لتكون قابلة للتوسيع بسهولة، وتدعم نماذج برمجة متعددة، بما في ذلك البرمجة الإجرائية (procedural)، والوظيفية (functional)، والشيئية (object-oriented).

تاريخ تطورها وأهم الإصدارات الرئيسية

- **أواخر الثمانينيات:** بدأ "Guido van Rossum" العمل على Python في مركز CWI بهولندا كخليفة للغة ABC.
- **1991:** تم نشر أول إصدار عام لـ Python (الإصدار 0.9.0).
- **1994:** صدر Python 1.0 مع ميزات جديدة مثل lambda, map, filter, و reduce.
- **2000:** صدر Python 2.0، والذي قدم ميزات مثل List Comprehensions ونظام جمع القمامة (garbage collection) يدعم الدورات المرجعية.
- **2008:** صدر Python 3.0 (المعروف أيضًا باسم Python 3000 أو Py3k). كان هذا إصدارًا رئيسيًا غير متوافق مع الإصدارات السابقة (backward-incompatible)، بهدف إصلاح العيوب الأساسية في تصميم اللغة وتبسيطها. **هذا هو الإصدار المستخدم والموصى به حاليًا.**
- **2020:** انتهى الدعم الرسمي لـ Python 2.7 في 1 يناير 2020، مما عزز التحول إلى Python 3.

فلسفة تصميم Python وأهم مبادئها (Zen of Python)

تتمثل فلسفة Python الأساسية في سهولة القراءة والكتابة. يمكن تلخيص مبادئها التوجيهية في "Zen of Python" (اكتب import this في مفسر Python لعرضها)، والتي تتضمن مبادئ مثل:

- الجميل أفضل من القبيح.
- الصريح أفضل من الضمني.
- البسيط أفضل من المعقد.
- المعقد أفضل من المعقد جدًا.
- القراءة مهمة.
- يجب أن يكون هناك طريقة واحدة - ويفضل أن تكون واضحة - للقيام بذلك.

أهميتها واستخداماتها الرئيسية

تُستخدم Python على نطاق واسع في مجالات متنوعة نظرًا لمرونتها ومكتباتها الغنية:

- **تطوير الويب (الخلفية):** باستخدام أطر عمل مثل Django و Flask و FastAPI.
- **علم البيانات وتحليل البيانات:** مع مكتبات مثل Pandas و NumPy و SciPy.
- **الذكاء الاصطناعي والتعلم الآلي:** باستخدام مكتبات قوية مثل TensorFlow و PyTorch و scikit-learn.
- **الأتمتة وكتابة النصوص البرمجية (Scripting):** لأتمتة المهام المتكررة وإدارة الأنظمة.
- **تطوير البرمجيات:** بناء تطبيقات سطح المكتب والأدوات المساعدة.
- **الحوسبة العلمية والهندسية:** بفضل مكتباتها المتخصصة.

- **تطوير الألعاب:** باستخدام مكتبات مثل Pygame.

2. بنية لغة Python الأساسية

أساسيات بناء الجملة

- **المسافات البادئة (Indentation):** تستخدم Python المسافات البادئة (عادة 4 مسافات) لتحديد كتل التعليمات البرمجية (بدلاً من الأقواس المعقوفة {} كما في لغات أخرى). المسافة البادئة إلزامية وجزء من بناء الجملة.
- **التعليمات (Statements):** عادة ما تنتهي التعليمة بنهاية السطر. يمكن فصل تعليمات متعددة في نفس السطر باستخدام الفاصلة المنقوطة (;)، ولكن هذا غير مستحسن.
- **التعليقات (Comments):** تبدأ التعليقات بعلامة الهاش (#) وتمتد حتى نهاية السطر. تُستخدم لشرح الكود.

```
# هذا تعليق
Python" = name" # هذا تعليق آخر في نفس السطر
print(name)

if True:
    print ("المسافة البادئة مهمة") # هذه الكتلة تابعة لـ if
```

أنواع البيانات الأساسية

- **الأرقام:**
  - int: الأعداد الصحيحة (مثل 10, -5, 0).
  - float: الأعداد العشرية (مثل 3.14, -0.5).
  - complex: الأعداد المركبة (مثل 3+5j).
- **القيم المنطقية:**
  - bool: تمثل True أو False.
- **النصوص:**
  - str: تسلسل من الأحرف (مثل "Hello", 'Python').
- **التسلسلات:**
  - list: قائمة قابلة للتغيير من العناصر (مثل [1, "apple", True]).
  - tuple: صف غير قابل للتغيير من العناصر (مثل (1, "apple", True)).
- **المجموعات:**
  - set: مجموعة غير مرتبة من العناصر الفريدة (مثل {1, "apple", True}).
- **التعيينات (Mapping):**
  - dict: قاموس (أو جدول هاش) يخزن أزواجًا من المفاتيح والقيم (مثل {"name": "Alice", "age": 30}).

المتغيرات

- لا تحتاج إلى تعريف نوع المتغير بشكل صريح في Python (الكتابة الديناميكية - Dynamic Typing). يتم تحديد نوع المتغير تلقائيًا عند إسناد قيمة له.
- يتم تعريف المتغير ببساطة عن طريق إسناد قيمة له باستخدام علامة يساوي (=).

```
age = 30          # age هو الآن int
price = 99.95     # price هو الآن float
is_active = True  # is_active هو الآن bool
message = "مرحباً"  # message هو الآن str
items = [1, 2, 3]  # items هو الآن list
```

عوامل التشغيل (Operators)

- **الحسابية:** + (جمع), - (طرح), * (ضرب), / (قسمة عشرية), // (قسمة صحيحة), % (باقي القسمة), ** (الأس).
- **المقارنة:** == (يساوي), != (لا يساوي), < (أكبر من), > (أصغر من), <= (أكبر من أو يساوي), >= (أصغر من أو يساوي).
- **المنطقية:** and (و), or (أو), not (ليس).
- **الإسناد:** = (إسناد), +=, -=, *=, /=, %= (إسناد مع عملية).
- **الهوية:** is (هل هو نفس الكائن؟), is not (هل ليس نفس الكائن؟).
- **العضوية:** in (هل العنصر موجود في التسلسل؟), not in (هل العنصر غير موجود في التسلسل؟).

3. التحكم في التدفق (Flow Control)

if/elif/else

تُستخدم لاتخاذ القرارات بناءً على شروط معينة.

```
score = 75

if score >= 90:
    print("ممتاز")
elif score >= 70:
    print("جيد جداً")
elif score >= 50:
    print("مقبول")
else:
    print("راسب")
```

for

تُستخدم للتكرار على عناصر تسلسل (مثل قائمة، صف، نص) أو أي كائن قابل للتكرار (iterable).

```
# التكرار على قائمة
fruits = ["تفاح", "موز", "برتقال"]
for fruit in fruits:
    print(fruit)
```

```
# التكرار باستخدام range()
for i in range(5): # من 0 إلى 4
    print(i)

# التكرار مع الفهرس
for index, fruit in enumerate(fruits):
    print(f"{index}: {fruit} الفهرس")
```

while

تُستخدم لتكرار كتلة من الكود طالما أن شرطًا معينًا يتحقق (True).

```
count = 0
while count < 3:
    print(f"العدد: {count}")
    count += 1
```

break, continue, pass

- break: تخرج فورًا من الحلقة الحالية (for أو while).
- continue: تتخطى باقي الكود في التكرار الحالي وتنتقل إلى التكرار التالي.
- pass: هي تعليمة فارغة، لا تفعل شيئًا. تُستخدم كعنصر نائب (placeholder) حيث يتطلب بناء الجملة وجود تعليمة ولكن لا يلزم تنفيذ أي إجراء.

```
for num in range(10):
    if num == 5:
        break # توقف عند 5
    if num % 2 == 0:
        continue # تخطى الأرقام الزوجية
    if num == 1:
        pass # لا تفعل شيئًا عند 1
    print(num) # سيطبع 3, 7, 9
```

4. الدوال (Functions) في Python

تعريف الدوال واستدعاؤها

تُستخدم الدوال لتجميع كود يمكن إعادة استخدامه. تُعرَّف باستخدام الكلمة المفتاحية def.

```
def greet(name):
    """هذه دالة ترحيب بسيطة."""
    print(f"مرحباً, {name}!")

# استدعاء الدالة
greet("علي")
```

معاملات الدوال (Arguments)

- **موضعية (Positional):** تُمرر بالترتيب.
- **مفتاحية (Keyword):** تُمرر باستخدام اسم المعامل (name=value).
- **قيم افتراضية (Default):** تُعطى قيمة افتراضية في تعريف الدالة.
- ***args**: لجمع عدد غير محدد من المعاملات الموضعية في tuple.
- ****kwargs**: لجمع عدد غير محدد من المعاملات المفتاحية في dict.

```
def describe_pet(pet_name, animal_type="كلب"): # animal_type له قيمة افتراضية
    print(f"لدي {animal_type} اسمه {pet_name}.")

describe_pet("ويلي") # استخدام القيمة الافتراضية
describe_pet(animal_type="قطة", pet_name="ميشو") # استخدام المعاملات المفتاحية

def sum_all(*numbers):
    total = 0
    for num in numbers:
        total += num
    return total

print(sum_all(1, 2, 3, 4)) # Output: 10

def print_info(**info):
    for key, value in info.items():
        print(f"{key}: {value}")

print_info(name="سارة", age=25, city="الرياض")
```

القيم المرجعة (return)

تُستخدم لإرجاع قيمة من الدالة. إذا لم تُستخدم return أو استُخدمت بدون قيمة، تُرجع الدالة None.

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

```
    return a + b

result = add(5, 3)
print(result) # Output: 8
```

الدوال المجهولة (Lambda Functions)

هي دوال صغيرة ومجهولة تُعرَّف باستخدام lambda. غالبًا ما تُستخدم كوسيط لدوال أخرى.

```
multiply = lambda x, y: x * y
print(multiply(4, 5)) # Output: 20

# استخدامها مع map
numbers = [1, 2, 3, 4]
squares = list(map(lambda x: x**2, numbers))
[print(squares) # Output: [1, 4, 9, 16
```

5. هياكل البيانات (Data Structures)

القوائم (Lists)

- تسلسل مرتب وقابل للتغيير (mutable) من العناصر.
- تُعرَّف باستخدام الأقواس المربعة [].
- **العمليات:**
  - append(item): إضافة عنصر للنهاية.
  - insert(index, item): إدراج عنصر في موضع معين.
  - remove(item): حذف أول ظهور للعنصر.
  - pop(index=-1): حذف وإرجاع العنصر في الموضع (الافتراضي هو الأخير).
  - index(item): إرجاع فهرس أول ظهور للعنصر.
  - count(item): إرجاع عدد مرات ظهور العنصر.
  - sort(): فرز القائمة في مكانها.
  - reverse(): عكس ترتيب القائمة في مكانها.
  - الوصول للعناصر: my_list[index].
  - التقطيع (Slicing): my_list[start:stop:step].

```
my_list = [1, "a", 3.14, "a"]
my_list.append(True)
print(my_list[1])  # Output: a
print(my_list.count("a")) # Output: 2
my_list.sort() # يسبب خطأ لأن العناصر من أنواع مختلفة لا يمكن مقارنتها مباشرة
```

```
# لنقم بفرز قائمة أرقام
num_list = [3, 1, 4, 1, 5, 9]
num_list.sort()
[print(num_list) # Output: [1, 1, 3, 4, 5, 9
```

الصفوف (Tuples)

- تسلسل مرتب وغير قابل للتغيير (immutable) من العناصر.
- تُعرَّف باستخدام الأقواس الهلالية ().
- بمجرد إنشائها، لا يمكن تعديل عناصرها أو إضافة أو حذف عناصر.
- أسرع بشكل عام من القوائم وتُستخدم غالبًا لتمثيل مجموعات ثابتة من البيانات.

```
my_tuple = (1, "a", 3.14)
print(my_tuple[0]) # Output: 1
TypeError سيسبب خطأ # # my_tuple[0] = 5
```

المجموعات (Sets)

- مجموعة غير مرتبة وغير مفهرسة من العناصر الفريدة.
- تُعرَّف باستخدام الأقواس المعقوفة {} أو الدالة set(). لإنشاء مجموعة فارغة، استخدم set() وليس {} (لأن {} تُنشئ قاموسًا فارغًا).
- تُستخدم للتحقق من العضوية بكفاءة وإزالة التكرارات وإجراء عمليات المجموعات الرياضية.
- **العمليات:**
  - add(item): إضافة عنصر.
  - remove(item): حذف عنصر (يُسبب خطأ إذا لم يكن موجودًا).
  - discard(item): حذف عنصر (لا يُسبب خطأ إذا لم يكن موجودًا).
  - pop(): حذف وإرجاع عنصر عشوائي.
  - union(other_set) أو |: الاتحاد.
  - intersection(other_set) أو &: التقاطع.
  - difference(other_set) أو -: الفرق.
  - issubset(other_set) أو <=: هل هي مجموعة جزئية؟
  - issuperset(other_set) أو >=: هل هي مجموعة شاملة؟

```
set1 = {1, 2, 3, 3, 4}
set2 = {3, 4, 5, 6}
{print(set1) # Output: {1, 2, 3, 4 (التكرار أُزيل، الترتيب غير مضمون)
print(set1.union(set2)) # Output: {1, 2, 3, 4, 5, 6}
{print(set1.intersection(set2)) # Output: {3, 4
```

القواميس (Dictionaries)

- مجموعة غير مرتبة (في Python < 3.7، مرتبة حسب الإدراج في Python >= 3.7) من أزواج key: value.
- المفاتيح يجب أن تكون فريدة وغير قابلة للتغيير (عادة أرقام، نصوص، أو صفوف).
- تُعرَّف باستخدام الأقواس المعقوفة {} مع أزواج المفتاح-القيمة.
- **العمليات:**
  - الوصول للقيمة: my_dict[key] (يُسبب خطأ KeyError إذا لم يكن المفتاح موجودًا).
  - get(key, default=None): الوصول للقيمة (يُرجع القيمة الافتراضية إذا لم يكن المفتاح موجودًا).
  - إضافة/تعديل: my_dict[key] = value.
  - pop(key, default=None): حذف وإرجاع القيمة المرتبطة بالمفتاح.
  - keys(): إرجاع عرض (view) للمفاتيح.
  - values(): إرجاع عرض للقيم.
  - items(): إرجاع عرض لأزواج (مفتاح، قيمة).

```
student = {"name": "خالد", "age": 20, "major": "هندسة"}
print(student["name"]) # Output: خالد
print(student.get("city", "غير محدد")) # Output: غير محدد
student["age"] = 21 # تعديل القيمة
student["gpa"] = 3.5 # إضافة زوج جديد
print(student.items()) # Output: dict_items([('name', 'خالد'), ('age', 21), ('major', 'هندسة'), ('gpa', 3.5')])
```

6. الوحدات (Modules) والحزم (Packages)

الوحدات (Modules)

- ملف Python (.py) يحتوي على تعريفات وتعليمات Python.
- تُستخدم لتنظيم الكود في أجزاء منطقية قابلة لإعادة الاستخدام.
- تُستورد باستخدام import module_name. يمكن الوصول إلى محتوياتها باستخدام module_name.function_name.
- يمكن استيراد أجزاء معينة باستخدام from module_name import specific_item أو from module_name import * (غير مستحسن بشكل عام).
- يمكن إعطاء اسم بديل للوحدة عند الاستيراد: import module_name as alias.

```
# استيراد وحدة math القياسية
import math
print(math.sqrt(16)) # Output: 4.0

# استيراد دالة محددة
from math import pi
print(pi) # Output: 3.14159...
```

```
# استيراد باسم بديل
import datetime as dt
today = dt.date.today()
(print(today
```

الحزم (Packages)

- طريقة لتنظيم وحدات Python ذات الصلة في هيكل مجلدات.
- الحزمة هي مجلد يحتوي على وحدات Python وملف خاص باسم __init__.py (يمكن أن يكون فارغًا)، والذي يُعلِم Python بأن هذا المجلد يجب أن يُعامل كحزمة.
- تسمح باستخدام التسمية النقطية للوحدات (dot notation)، مثل package_name.module_name.

وحدات Python القياسية الشائعة

- math: دوال رياضية.
- os: التفاعل مع نظام التشغيل (ملفات، مجلدات، عمليات).
- sys: الوصول إلى متغيرات ومؤشرات خاصة بالمفسر.
- datetime: التعامل مع التواريخ والأوقات.
- json: التعامل مع بيانات بصيغة JSON.
- random: توليد أرقام عشوائية.
- re: التعامل مع التعابير النمطية (Regular Expressions).

مدير الحزم pip

- الأداة القياسية لتثبيت وإدارة الحزم الخارجية (المكتبات) التي يطورها مجتمع Python.
- **الأوامر الشائعة:**
  - pip install package_name: تثبيت حزمة.
  - pip uninstall package_name: إلغاء تثبيت حزمة.
  - pip list: عرض الحزم المثبتة.
  - pip freeze > requirements.txt: حفظ قائمة الحزم المثبتة وإصداراتها في ملف.
  - pip install -r requirements.txt: تثبيت الحزم من ملف.

7. البرمجة الشيئية (Object-Oriented Programming - OOP)

المفاهيم الأساسية

- **الأصناف (Classes):** هي المخططات أو القوالب (blueprints) لإنشاء الكائنات. تُعرَّف باستخدام الكلمة المفتاحية class.
- **الكائنات (Objects):** هي نسخ (instances) من الأصناف. لكل كائن حالته (attributes) وسلوكياته (methods) الخاصة.
- **التغليف (Encapsulation):** تجميع البيانات (السمات) والعمليات التي تعمل عليها (الدوال/الطرق) داخل وحدة واحدة (الكائن). في Python، يتم تحقيق التغليف بشكل أساسي عن طريق الاتفاق (convention) باستخدام بادئات

مثل _ (محمي) أو __ (خاص - مع name mangling).

- **الوراثة (Inheritance):** آلية تسمح لصنف جديد (الصنف المشتق أو الابن) بأن يرث السمات والطرق من صنف موجود (الصنف الأساسي أو الأب). هذا يعزز إعادة استخدام الكود. تدعم Python الوراثة المتعددة (multiple inheritance).
- **تعدد الأوجه (Polymorphism):** قدرة الكائنات من أصناف مختلفة على الاستجابة لنفس الرسالة (استدعاء الدالة) بطرق مختلفة. في Python، غالبًا ما يتحقق هذا من خلال "Duck Typing" (إذا كان يمشي كالبطة ويصدر صوت البطة، فهو بطة) - لا يهم نوع الكائن بقدر ما يهم ما إذا كان يحتوي على الطرق أو السمات المطلوبة.

تعريف الأصناف والكائنات

```
class Dog:
    # سمة على مستوى الصنف (مشتركة بين كل الكائنات)
    species = "Canis familiaris"

    # المُنشئ (Initializer/Constructor)
    def __init__(self, name, age):
        # سمات على مستوى الكائن (خاصة بكل كائن)
        self.name = name
        self.age = age
        self._secret = "هذا سر" # سمة محمية (اتفاق)
        self.__private_stuff = "خاص جداً" # سمة خاصة (name mangling)

    # طريقة (Method) على مستوى الكائن
    def bark(self):
        print(f"{self.name} يقول: Woof!")

    def get_private(self):
        return self.__private_stuff

# إنشاء كائنات (نسخ) من الصنف Dog
my_dog = Dog("Buddy", 3)
your_dog = Dog("Lucy", 5)

# الوصول إلى السمات واستدعاء الطرق
print(my_dog.name)      # Output: Buddy
print(your_dog.age)     # Output: 5
print(Dog.species)      # Output: Canis familiaris
my_dog.bark()           # Output: Buddy يقول: Woof!
print(my_dog._secret)   # يمكن الوصول إليه (لكنه اتفاق على عدم القيام بذلك مباشرة)
# print(my_dog.__private_stuff) # سيسبب خطأ AttributeError
```

```
print(my_dog.get_private()) # Output: خاص جداً
print(my_dog._Dog__private_stuff) # الوصول عبر name mangling (غير مستحسن)

# مثال على الوراثة
class Bulldog(Dog): # Bulldog يرث من Dog
    def __init__(self, name, age, weight):
        super().__init__(name, age) # استدعاء مُنشئ الصنف الأب
        self.weight = weight

    # تجاوز طريقة الأب (Overriding)
    def bark(self):
        print(f"{self.name} (بولدوج) يقول: Grrr-Woof!")

my_bulldog = Bulldog("Rocky", 4, 25)
my_bulldog.bark() # Output: Rocky (بولدوج) يقول: Grrr-Woof!
print(my_bulldog.age) # Output: 4 (ورثها من Dog)
```

الدوال الخاصة (Special/Magic Methods)

هي دوال محاطة بشرطتين سفليتين مزدوجتين (__method__) ولها معنى خاص في Python.

- __init__(self, ...): المُنشئ، يُستدعى عند إنشاء كائن جديد.
- __str__(self): يُرجع تمثيلًا نصيًا "غير رسمي" للكائن (يُستخدم بواسطة print() و str()).
- __repr__(self): يُرجع تمثيلًا نصيًا "رسميًا" للكائن (يُستخدم للمطورين والتصحيح).
- __len__(self): يُرجع طول الكائن (يُستخدم بواسطة len()).
- __add__(self, other): يحدد سلوك عامل الجمع +.

8. التعامل مع الملفات (File Handling)

فتح وقراءة وكتابة الملفات

- تُستخدم الدالة open(filepath, mode) لفتح ملف وإرجاع كائن ملف.
- من الأفضل دائمًا استخدام عبارة with open(...) as file: لأنها تضمن إغلاق الملف تلقائيًا حتى لو حدثت أخطاء.

```
# الكتابة إلى ملف (سيتم إنشاء الملف إذا لم يكن موجودًا، أو الكتابة فوقه إذا كان موجودًا)
try:
    with open("my_file.txt", "w", encoding="utf-8") as f:
        f.write("هذه هي السطر الأول.\n")
        f.write("وهذا هو السطر الثاني.\n")
    print("تمت الكتابة إلى الملف بنجاح.")
except IOError as e:
    print(f"حدث خطأ أثناء الكتابة: {e}")
```

```
# القراءة من ملف
try:
    with open("my_file.txt", "r", encoding="utf-8") as f:
        # قراءة الملف بالكامل
        # content = f.read()
        # print("محتوى الملف بالكامل:")
        # print(content)

        # قراءة الملف سطراً بسطر
        print("\nقراءة الملف سطراً بسطر:")
        for line in f:
            print(line.strip()) # .strip() لإزالة مسافات البداية/النهاية وسطر جديد

        # الانتقال إلى بداية الملف مرة أخرى للقراءة بطريقة أخرى
        f.seek(0)

        # قراءة جميع الأسطر في قائمة
        lines = f.readlines()
        print("\nقائمة الأسطر:")
        print(lines)

except FileNotFoundError:
    print("خطأ: الملف غير موجود.")
except IOError as e:
    print(f"حدث خطأ أثناء القراءة: {e}")

# الإلحاق بملف (إضافة محتوى إلى نهاية الملف)
try:
    with open("my_file.txt", "a", encoding="utf-8") as f:
        f.write("هذا سطر إضافي.\n")
    print("\nتم إلحاق المحتوى بالملف.")
except IOError as e:
    print(f"حدث خطأ أثناء الإلحاق: {e}")
```

أوضاع فتح الملفات

- 'r': قراءة (افتراضي).
- 'w': كتابة (الكتابة فوق الملف الموجود أو إنشاء ملف جديد).
- 'a': إلحاق (إضافة إلى نهاية الملف).
- 'b': وضع ثنائي (للتعامل مع الملفات غير النصية مثل الصور).
- '+': فتح للتحديث (قراءة وكتابة). (مثل 'r+' أو 'w+')

9. التعامل مع الاستثناءات (Exception Handling)

try/except/finally

تُستخدم للتعامل مع الأخطاء (الاستثناءات) التي قد تحدث أثناء تنفيذ الكود.

- **try**: يحتوي على الكود الذي قد يثير استثناءً.
- **except ExceptionType:**: يحتوي على الكود الذي يتم تنفيذه إذا حدث استثناء من النوع المحدد في كتلة try. يمكن وجود عدة كتل except لأنواع مختلفة من الاستثناءات.
- **else**: (اختياري) يحتوي على الكود الذي يتم تنفيذه إذا لم تحدث أي استثناءات في كتلة try.
- **finally**: (اختياري) يحتوي على الكود الذي يتم تنفيذه دائمًا، سواء حدث استثناء أم لا (مفيد لتنظيف الموارد مثل إغلاق الملفات).

```
try:
numerator = 10
(("أدخل المقام: ")) denominator = int(input
result = numerator / denominator
except ZeroDivisionError:
("خطأ: لا يمكن القسمة على صفر!") print
except ValueError:
("خطأ: يجب إدخال رقم صحيح.") print
except Exception as e: # التقاط أي استثناء آخر
print(f"حدث خطأ غير متوقع: {e}")
else:
print(f"النتيجة هي: {result}") 
finally:
("تنفيذ كتلة finally دائمًا.") print
```

أنواع الاستثناءات الشائعة

TypeError, ValueError, IndexError, KeyError, FileNotFoundError, ZeroDivisionError, AttributeError, ImportError.

رفع استثناءات (raise)

يمكنك إثارة استثناء بشكل صريح باستخدام الكلمة المفتاحية raise.

```
def set_age(age):
    if age < 0:
        raise ValueError("العمر لا يمكن أن يكون سالبًا.")
    print(f"تم تعيين العمر إلى: {age}")

try:
    set_age(-5)
except ValueError as e:
    print(f"خطأ في الإدخال: {e}")
```

10. مفاهيم متقدمة (نظرة عامة)

- **المولدات (Generators):** طريقة لإنشاء مُكرِرات (iterators) بكفاءة في استخدام الذاكرة. تستخدم الكلمة المفتاحية yield لإرجاع قيمة واحدة في كل مرة، مع الحفاظ على حالتها بين الاستدعاءات. مفيدة للتعامل مع تسلسلات كبيرة جدًا.
- **الزخارف (Decorators):** طريقة لتعديل أو تحسين الدوال أو الأصناف بطريقة نظيفة وقابلة لإعادة الاستخدام. هي في الأساس دوال تأخذ دالة أخرى كمدخل وتُرجع دالة معدلة. تُستخدم عادةً للتسجيل (logging)، التحكم في الوصول، التوقيت، وغيرها.
- **مديرو السياق (Context Managers):** تُستخدم لإدارة الموارد (مثل الملفات أو اتصالات الشبكة) عن طريق ضمان إعدادها وتفكيكها بشكل صحيح. تُستخدم عادةً مع عبارة with. يمكن إنشاؤها باستخدام الأصناف (مع __enter__ و __exit__) أو باستخدام الدالة contextlib.contextmanager مع مولد.
- **التزامن والتوازي (Concurrency and Parallelism):**
  - **threading**: لتشغيل عدة أجزاء من الكود بشكل متزامن داخل نفس العملية (مفيد للمهام المرتبطة بالإدخال/الإخراج). يواجه قيود GIL (Global Interpreter Lock) في CPython بالنسبة للمهام المرتبطة بوحدة المعالجة المركزية.
  - **multiprocessing**: لتشغيل عدة عمليات منفصلة، مما يسمح بالتوازي الحقيقي (تجاوز قيود GIL) للمهام المرتبطة بوحدة المعالجة المركزية.
  - **asyncio**: إطار عمل للبرمجة غير المتزامنة باستخدام async/await. فعال جدًا للمهام التي تتضمن انتظار عمليات إدخال/إخراج (مثل الشبكات) دون حظر الخيط الرئيسي.

تم إنتاج هذا الملخص بواسطة المبرمج SayyadN.