

Mindset: Introduction to AI

ملخص المحاضرة الثالثة -- AI

1 . Tuples (الصفوف)

تعريف الـ Tuple:

`(3 ,3 ,2 ,1) = my_tuple`

- الـ Tuples عبارة عن نوع من البيانات يشبه القوائم ولكنه غير قابل للتغيير. (Immutable)
- يمكن أن تحتوي على عناصر مكررة.
- تدعم أنواع بيانات مختلفة داخل نفس الـ Tuple.
- مرتبة (Ordered) ، أي يمكن الوصول للعناصر باستخدام الفهرس. (Indexing)

بعض العمليات على الـ Tuples:

```
dir(my_tuple) # عرض الدوال المتاحة
my_tuple.count(1) # Tuple عدد مرات تكرار العنصر 1 داخل الـ
my_tuple.index(1) # الحصول على الفهرس الأول لوجود العنصر 1
my_tuple[1] # Tuple الوصول للعنصر الثاني في الـ
# my_tuple[1] = 8 # غير قابلة للتغيير Tuple خطأ، لأن الـ
```

تخزين الـ Tuples داخل القواميس:

```
my_dic = {"mohamed": (1, 2, 3), "Bodor": (4, 5, 6)}
my_dic["mohamed"] # الوصول إلى القيم المرتبطة بالمفتاح
```



2 . Sets (المجموعات)

تعريف الـ Set:

set1 = {3, True, 5, 7}

- الـ **Sets** عبارة عن مجموعة غير مرتبة من العناصر الفريدة (لا تقبل التكرار).
- تدعم إضافة وإزالة العناصر لكنها غير مرتبة.

بعض العمليات عليها

```
dir(set) # عرض الدوال المتاحة
set1.add(3) # إضافة عنصر (إذا كان موجودًا فلن يتكرر)
set1.remove(1) # إزالة العنصر 1، إذا لم يكن موجودًا يحدث خطأ
set1.discard(3) # إزالة العنصر 3، وإذا لم يكن موجودًا لا يحدث خطأ
set1.clear() # حذف جميع العناصر من الـ Set
type(set1) # معرفة نوع البيانات
len(set1) # حساب عدد العناصر في الـ Set
```

أهم الفروقات بين Sets و Tuples

Sets	Tuples	الخاصية
لا	نعم	مرتبة؟
لا	نعم	تقبل التكرار؟
نعم	لا	قابلة للتغيير؟
لا	نعم	إمكانية الفهرسة؟

ملخص سريع

- الـ **Tuples** مرتبة، تقبل التكرار، لكنها غير قابلة للتغيير.
- الـ **Sets** غير مرتبة، لا تقبل التكرار، قابلة للتعديل.
- الفرق بين `remove()` و `discard()`:
 - `remove()` يحذف العنصر، وإذا لم يكن موجودًا يظهر خطأ.
 - `discard()` يحذف العنصر، وإذا لم يكن موجودًا لا يظهر خطأ.

3. مفهوم الـ Modules في بايثون

في حياتنا اليومية، نستخدم أجهزة مثل الكمبيوتر، الهاتف، والسيارة دون الحاجة إلى إعادة اختراعها، بل نعيد استخدامها.

ينطبق نفس المفهوم في البرمجة، حيث تسمح لنا الموديولات (Modules) بإعادة استخدام أكواد كتبها الآخرون بدلاً من كتابتها من الصفر.

الموديولات توفر:

إعادة استخدام الكود بسهولة.

تنظيم الكود وتقسيمه إلى أجزاء صغيرة قابلة للإدارة.

تقليل الأخطاء من خلال الاستفادة من أكواد موثوقة.

1. مكتبة math

🔗 الموديول math يوفر دوال رياضية جاهزة:

```
import math
math.pi          # (باي) قيمة  $\pi$ 
math.sqrt(16)     # الجذر التربيعي 4
math.ceil(1.2)    # التقريب للأعلى = 2
math.floor(1.2)   # التقريب للأسفل = 1
math.pow(2,3)     # الأس ( $3^2$ ) = 8
math.fabs(-11)    # القيمة المطلقة = 11
math.factorial(5) # مضروب العدد  $5! = 120$ 
math.gamma(5.1)   # دالة جاما، تعميم للمضروب (ن - 1)
math.log10(100)   # اللوغاريتم العشري = 2
```

🔗 **dir(math):** يعرض جميع الدوال داخل .

2. مكتبة calendar

🔗 الموديول calendar للتعامل مع التواريخ والتقويمات:

```
import calendar
print(calendar.month(2022,11)) # طباعة تقويم شهر نوفمبر 2022

calendar.isleap(2022) # التحقق مما إذا كانت سنة كبيسة (False)
```

💡 `dir(calendar)` : يعرض جميع الدوال داخل `calendar` .

3 . مكتبة `statistics`

🔗 الموديول `statistics` للتحليل الإحصائي:

```
import statistics
statistics.mean([1,2,3,4,5]) # المتوسط الحسابي = 3.0
statistics.median([1,2,3,4,5]) # الوسيط = 3
statistics.mode([1,2,3,4,5]) # المنوال = 1 (أول عنصر مكرر)
```

💡 `dir(statistics)` : يعرض جميع الدوال داخل `statistics`

💡 خلاصة المحاضرة

الموديول	الوظيفة الأساسية
<code>math</code>	عمليات رياضية متقدمة
<code>calendar</code>	التعامل مع التواريخ والتقويمات
<code>statistics</code>	العمليات الإحصائية

- ♦ الموديولات تجعل البرمجة أسهل من خلال إعادة استخدام أكواد جاهزة وموثوقة!
- ♦ استخدم `dir(اسم_الموديول)` لمعرفة جميع الدوال المتاحة داخله.

شرح Counter Module في بايثون

1. ما هو Counter؟

- Counter هو جزء من collections في بايثون، وهو نوع بيانات خاص يستخدم لحساب تكرار العناصر في القوائم، السلاسل النصية، القواميس، وغيرها.
- يشبه القاموس (dict) لكنه مخصص لحساب التكرارات.

كيفية استيراد الموديول:

```
from collections import Counter
```

2. إنشاء Counter بطرق مختلفة

Edit Copy

```
c1 = Counter("mississippi") # من سلسلة نصية
c2 = Counter({'a': 3, 'b': 1}) # من قاموس
c3 = Counter(a=2, b=3, c=1) # باستخدام `keyword arguments`

print("Counter from iterable:", c1)
print("Counter from dict:", c2)
print("Counter from keyword args:", c3)
```

المخرجات: ◆

Edit Copy

```
Counter from iterable: Counter({'i': 4, 's': 4, 'p': 2, 'm': 1})
Counter from dict: Counter({'a': 3, 'b': 1})
Counter from keyword args: Counter({'b': 3, 'a': 2, 'c': 1})
```

3. دوال هامة في Counter

♦ elements() - إرجاع العناصر وفقًا لتكرارها

Edit Copy

```
c_elements = list(Counter({'a': 3, 'c': 1}).elements())
print("Elements:", c_elements)
```

💡 المخرجات:

Edit Copy

```
Elements: ['a', 'a', 'a', 'c']
```

♦ most_common(n) - إرجاع أكثر العناصر تكرارًا

Edit Copy

```
c_most_common = Counter("banana").most_common(2)
print("Most Common:", c_most_common)
```

💡 المخرجات:

Edit Copy

```
Most Common: [('a', 3), ('n', 2)]
```

♦ subtract() - الطرح بين التكرارات

Edit Copy



```
c_subtract = Counter(a=4, b=2, c=0)
c_subtract.subtract({'a': 2, 'b': 1, 'c': 1})
print("After Subtract:", c_subtract)
```

💡 المخرجات:

Edit Copy

```
After Subtract: Counter({'a': 2, 'b': 1, 'c': -1})
```

♦ update () - تحديث القيم بإضافة تكرارات جديدة

Edit  Copy 

```
c_update = Counter({'a': 2, 'b': 1})  
c_update.update('abbc')  
print("After Update:", c_update)
```

💡 المخرجات:

Edit  Copy 

```
After Update: Counter({'a': 3, 'b': 3, 'c': 1})
```

4. العمليات الحسابية على Counter

```
c1_op = Counter(a=3, b=1)  
c2_op = Counter(a=1, b=2, c=1)  
  
c_add = c1_op + c2_op    # جمع التكرارات  
c_sub = c1_op - c2_op    # طرح التكرارات  
c_intersect = c1_op & c2_op # التقاطع (القيم الصغرى)  
c_union = c1_op | c2_op  # الاتحاد (القيم الكبرى)  
  
print("Addition:", c_add)  
print("Subtraction:", c_sub)  
print("Intersection:", c_intersect)  
print("Union:", c_union)
```

المخرجات:

Addition: Counter({'a': 4, 'b': 3, 'c': 1})

Subtraction: Counter({'a': 2})

Intersection: Counter({'a': 1, 'b': 1})

Union: Counter({'a': 3, 'b': 2, 'c': 1})

5. وظائف أخرى في Counter

clear() - تفريغ جميع البيانات

```
c_clear = Counter(a=2, b=1)
c_clear.clear()
print("After Clear:", c_clear)
```

Edit Copy

المخرجات:

```
After Clear: Counter()
```

Edit Copy

keys(), values(), items() - استخراج المفاتيح، القيم، والعناصر

```
c_keys = Counter("apple").keys()
c_values = Counter("apple").values()
c_items = Counter("apple").items()

print("Keys:", list(c_keys))
print("Values:", list(c_values))
print("Items:", list(c_items))
```

Edit Copy

المخرجات:

```
Keys: ['a', 'p', 'l', 'e']
Values: [1, 2, 1, 1]
Items: [('a', 1), ('p', 2), ('l', 1), ('e', 1)]
```

Edit Copy

تحويل Counter إلى Dictionary

Edit Copy

```
c_dict = dict(Counter("hello"))
print("Counter to Dict:", c_dict)
```

المخرجات:

Edit Copy

```
Counter to Dict: {'h': 1, 'e': 1, 'l': 2, 'o': 1}
```

حذف عنصر من Counter

Edit Copy

```
c_delete = Counter(a=3, b=2)
del c_delete['a']
print("After Delete:", c_delete)
```

المخرجات:

Edit Copy

```
After Delete: Counter({'b': 2})
```

6. تطبيق عملي: حساب تكرار الكلمات في نص

Edit Copy

```
text = "apple banana apple orange banana apple"
words = text.split()
word_counts = Counter(words)
word_common = word_counts.most_common(2)

print("Word Frequency:", word_common)
```

المخرجات:

Edit Copy

```
Word Frequency: [('apple', 3), ('banana', 2)]
```

ملخص بشكل عام:

الوصف	الوظيفة
إنشاء Counter من نص، قائمة أو قاموس	<code>Counter(iterable)</code>
إرجاع العناصر مع تكرارها	<code>()elements.</code>
إرجاع أكثر n عناصر تكرارًا	<code>most_common(n).</code>
طرح التكرارات	<code>subtract(Counter).</code>
إضافة عناصر وتحديث التكرارات	<code>update(iterable).</code>
حذف جميع القيم	<code>()clear.</code>
استخراج المفاتيح، القيم، والعناصر	<code>() keys(), .values(), .items .</code>
جمع تكرارات	<code>Counter + Counter</code>
طرح التكرارات	<code>Counter - Counter</code>
التقاطع (القيم الصغرى)	<code>Counter & Counter</code>
<code>`Counter</code>	<code>Counter`</code>
حذف عنصر	<code>del Counter['key']</code>

Counter هو أداة قوية لحساب التكرارات، ويمكن استخدامه بسهولة في تحليل النصوص، الإحصائيات، ومعالجة البيانات!

وظائف مكتبة random وitertools في بايثون:

مكتبة random (لتوليد الأرقام العشوائية)

1. `seed(x)` – يحدد نقطة البداية للمولد العشوائي لضمان نفس النتائج كل مرة.
2. `getstate()` – يسترجع الحالة الداخلية الحالية للمولد العشوائي.
3. `setstate(state)` – يستعيد حالة المولد العشوائي لضمان نفس النتائج.
4. `getrandbits(n)` – يولد عددًا صحيحًا عشوائيًا بعدد n من البتات.

5. `randrange(start, stop, step)` – يولد عددًا عشوائيًا في مدى معين (الحد العلوي غير مشمول).
6. `randint(a, b)` – يولد عددًا صحيحًا عشوائيًا بين `a` و `b` (بشمولين).
7. `choice(seq)` – يختار عنصرًا عشوائيًا من قائمة.
8. `sample(seq, k=n)` – يختار `n` عناصر فريدة من القائمة.
9. `random()` – يولد عددًا عشويًا عشوائيًا بين 0 و 1.
10. `uniform(a, b)` – يولد عددًا عشويًا عشوائيًا بين `a` و `b`.
11. `triangular(low, high, mode)` – يولد عددًا عشويًا عشوائيًا بانحياز نحو `mode`.
-

مكتبة `itertools` (للتكرارات والتعامل مع المجموعات)

✓ المولدات اللانهائية

- `count(start, step)` – يولد أرقامًا متتالية تبدأ من `start` بمقدار `step`.
- `cycle(iterable)` – يعيد تكرار عناصر القائمة إلى ما لا نهاية.
- `repeat(item, times)` – يكرر عنصرًا معينًا عددًا محددًا من المرات.

✓ دمج وترتيب القوائم

- `chain(*iterables)` – يربط عدة قوائم معًا كسلسلة واحدة.
- `zip(*iterables)` – يدمج عناصر القوائم في `tuples` حتى أقصر قائمة.
- `zip_longest(*iterables, fillvalue)` – مثل `zip` ولكنه يملأ القيم الناقصة.
- `product(*iterables)` – ينتج جميع التوليفات الممكنة بين القوائم (الضرب الديكارتي).

✓ التصفية والتقسيم

- `islice(iterable, start, stop, step)` – يقطع جزءًا من القائمة مثل `slice()`.
- `takewhile(predicate, iterable)` – يأخذ العناصر طالما تحقق الشرط.
- `dropwhile(predicate, iterable)` – يحذف العناصر حتى يفشل الشرط لأول مرة.
- `filterfalse(predicate, iterable)` – يحتفظ بالعناصر التي لا تحقق الشرط.

✓ التجميع والتكرار

- `groupby(iterable, key)` – يجمع القيم المتشابهة بناءً على مفتاح معين.
- `tee(iterable, n)` – ينشئ `n` نسخ مستقلة من نفس المولد.

🔗 Functions Summary

1 Default Parameters

```
def sum(x=0, y=2): # Default values for x and y
    return x + y

print(sum(5, 3)) # Output: 8
```

✓ إذا لم يُمرَّر أي قيمة للمعاملات، سنستخدم القيم الافتراضية $x=0$ و $y=2$.

2 Function with Default and Named Arguments

```
def cylinder_volume(radius, height=1):
    print("radius is:", radius)
    print("height is:", height)
    return 3.14 * (radius ** 2) * height

r = 5
h = 10
print(cylinder_volume(height=h, radius=r)) # Named arguments
print(cylinder_volume(r, h)) # Positional arguments
```

✓ يمكن استخدام **arguments المسماة** لتحديد القيم بوضوح.

✓ المعامل `height` لديه قيمة افتراضية 1.

3 Basic Function Without Parameters

```
def greet():  
    print('Hello World!')  
  
greet()  
print('Outside function')
```

✓ دالة بدون معاملات تطبع رسالة.

✓ الدالة تُستدعى مباشرةً باستخدام . greet()

4 Function with a Single Parameter

```
def greet(name):  
    print("Hello", name)  
  
greet("John")
```

✓ تمرير argument واحد عند استدعاء الدالة.

5 Function with Multiple Parameters

```
def add_numbers(num1, num2):  
    sum = num1 + num2  
    print("Sum: ", sum)
```

```
add_numbers(5, 4) # Output: Sum: 9
```

✓ تأخذ الدالة قيمتين وتطبع ناتج جمعهما.

6 Function with Return Value

```
def find_square(num):  
    return num * num  
  
square = find_square(3)  
print('Square:', square) # Output: Square: 9
```

✓ تُعيد الدالة ناتج مربع العدد.

📌 Local and Global Variables Summary

1 Local Variable Example

```
def f():  
    # Local variable (only exists inside the function)  
    s = "I love Geeksforgeeks"  
    print(s)  
  
# Calling the function  
f()
```

✓ هنا متغير محلي s لا يمكن الوصول إليه خارج الدالة.

2 Accessing a Global Variable Inside a Function

```
# Global variable  
s = "I love Geeksforgeeks"  
  
def f():  
    print("Inside Function:", s) # Accessing global variable  
  
f()  
print("Outside Function:", s)
```

✓ يمكن للدالة الوصول إلى المتغيرات العامة التي تم تعريفها خارجها.

3 Local Variable with Same Name as Global Variable

```
s = "I love Geeksforgeeks" # Global variable

def f():
    s = "Me too." # Local variable (shadows the global variable)
    print(s)

f()
print(s) # Global variable remains unchanged
```

إذا تم تعريف متغير داخل الدالة بنفس اسم المتغير العام، فإنه لن يؤثر على المتغير العام. ✓

4 Global and Local Scope with Same Variable Name

```
a = 1 # Global variable

def f():
    print('Inside f() :', a) # Uses global variable

def g():
    a = 2 # Local variable (does not affect the global one)
    print('Inside g() :', a)

def h():
    global a # Modifies the global variable
    a = 3
    print('Inside h() :', a)

print('Global:', a)
f() # Uses global 'a'
print('Global:', a)
g() # Uses local 'a' inside g()
print('Global:', a) # Global 'a' remains unchanged
h() # Modifies global 'a'
print('Global:', a) # Now global 'a' is changed
f() # Reflects the change
```

إذا لم يتم تعريف متغير محلي داخل الدالة، فسيتم استخدام المتغير العام. ✓

استخدام global a داخل الدالة يجعلها تغير المتغير العام مباشرة. ✓

